Graphics • Windows • ASM • C • C++ • BASIC • COBOL • SQL

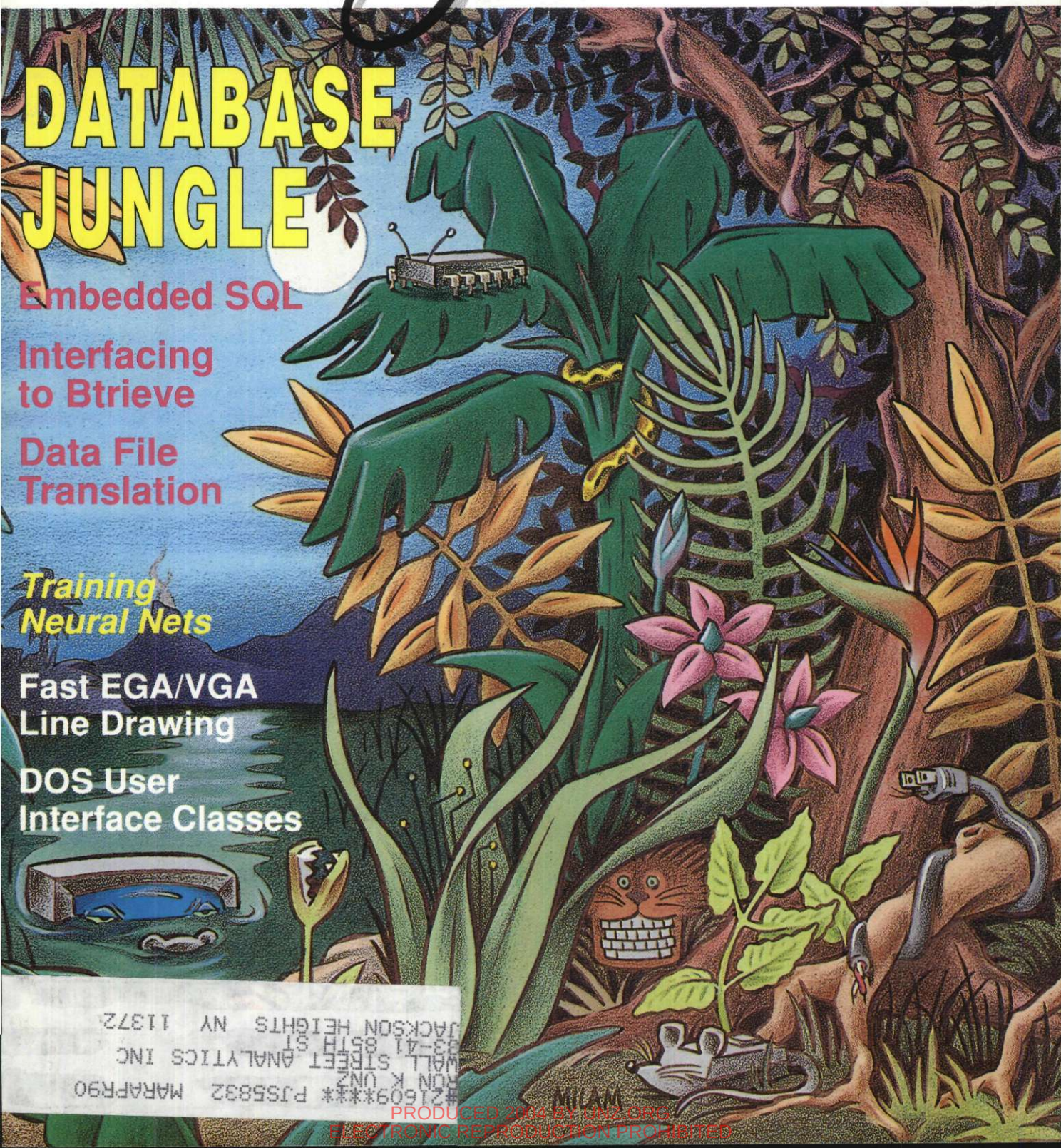
The Resource Journal for IBM Micro Programmers

PROGRAMMER'S Journal™

SQL FEATURE

Windows FEATURE

MAY/JUNE 1989
Volume 7.3
$4.95 Canada $5.95



DATABASE JUNGLE

Embedded SQL

Interfacing to Btrieve

Data File Translation

Training Neural Nets

Fast EGA/VGA Line Drawing

DOS User Interface Classes

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Now it's easy to make your programs millions of times smarter.

More and more, programmers and work station builders are using DESQview as a development tool. The reason is simple. They can create powerful, multitasking solutions today for the millions of DOS PCs in use today. Solutions comparable to those promised for tomorrow by OS/2.

The API Advantage

Programmers who take advantage of DESQview's API (Application Program Interface) get access to the powerful capabilities built into DESQview —multitasking, windowing, intertask communications, mailboxes, shared programs, memory management, mousing, data transfer, menu-building and context sensitive help.

Bells and Whistles

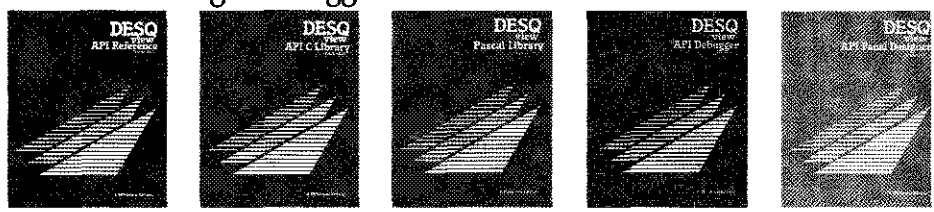
A program taking advantage of the DESQview API can spawn subtasks for performing background operations or new processes for loading and running other programs concurrently. It can schedule processing after an interval or at a certain time. It can use DESQview's intertask communications to rapidly exchange data between programs, share common code and data; or interrupt at critical events. It can use DESQview's menuing and mousing capabilities to create menus. And there's lots more it can do.

Some of the applications under development right now using DESQview API Tools: CAD, medical systems, insurance, 3270 mainframe communications, network management, real estate, typesetting, point of sale, education, commodity trading, stock trading and online voting.

80386 Power

Programmers can take advantage of the 80386's protected mode for large programs, yet run on DOS and multitask in DESQview—side by side with other 80386 and DOS programs. The breakthroughs that make this possible: DOS Extenders from PharLap Software and AI Architects and DESQview support of these DOS extenders.

A Complete Interactive Toolkit —Including a Debugger



Quarterdeck can provide everything you need to put DESQview capabilities into your applications.

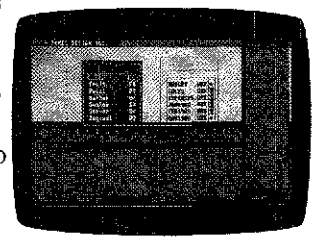
API Reference Manual The key to the power of the DESQview API, this manual contains all you need to know to write Assembly Language programs that take full advantage of DESQview's capabilities. And there's an 'include' file with symbols and macros to aid you in development.

API C Library Here are C language interfaces for the entire set of API functions. It supports the Lattice™ C, Metaware™ C, Microsoft® C, and Turbo C compilers for all memory models. Included with the C Library package is the API Reference Manual and source code for the library.

API Pascal Library The Pascal library provides interfaces for the entire set of API functions. It supports Turbo Pascal V4.0 and V5.0 compilers. Included are the API Reference Manual, source code for the library, and example programs.

API Debugger The DESQview API Debugger is an interactive tool enabling the API programmer to trace and single step through API calls from several concurrently running DESQview-specific programs. Trace information is reported symbolically along with the program counter,registers, and stack at the time of the call. Trace conditions can be specified to report only calls of interest.

API Panel Designer This interactive tool help you design windows, menus, help screens, error messages, and forms. It includes an editor that let you construct an image of your panel using simpl commands to enter, edit, copy, and move text, as well as draw lines and boxes. You can then define the characteristics of the window that will contain the panel, such as its position, size, and title. Finally, you can specify the locations and types of fields in the panel.



The Panel Designer automatically generates all the DESQview API da streams necessary display and take input from your panel. These data streams may be grouped into panel libraries and stored on disk or as part of your program.

More Tools are Coming Quarterdeck is committe to adding tools as needed by our users. To that en we have been working with Ashton Tate and Buz words International on dBASE III and dBASE IV translators. And in the works, we have BASIC an DOS Extender libraries.

If you'd like hands-on experience with the API Toolkit, write for information on our next API Conference, coming this August.

Bring New Power to DOS with the DESQview API Toolkit.

Quarterdeck

Quarterdeck Office Systems, 150 Pico Blvd., Santa Monica, CA 90405 (213) 392-9851 Fax: (213) 399-3802

Number 147 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Simply the BEST C and Pascal on AT, 386, Sun, Apollo, RT, VAX, 370

"The most rock-solid C compiler in the industry. Superb technical support and portability. Superior code generated."
Gordon Eubanks, Symantec — Q&A (386).

"It simply works, with no trouble, no chasing strange bugs, and excellent warning and error messages ... a professional product."
Robert Lerche, Bay Partners.

"For large-scale software development, the highest quality C compiler available on the market today. Pragmas are great. Quality of support is exceptional." **Randy Neilsen, Ansa—Paradox** (DOS,OS/2).

"15% smaller and 15% faster than Lattice C."
Robert Wenig, Autodesk.

"Our software is running anywhere from 30 to 50% faster than when compiled under Lattice." **David Marcus, Micronetics.**

"We switched from Lattice due to a 10% reduction in code size. The compiler is very stable." **Lee Lorenzen, Ventura Software — Ventura Publisher, marketted by Xerox Corp.**

"Best quality emitted code by any compiler I've encountered. Often amazing." **Bill Ferguson, Fox Software — FoxBase (386).**

"Messages sometimes pointed out type mismatches, incorrect-length argument lists, and uninitialized variables that had been undetected for years [in 4.x bsd]." **Larry Breed, IBM ACIS [RT PC].**

"Diagnostics turned up bugs missed by other compilers. Rapid bug fixes by technical support, someone who knew what he was talking about. 80386 code is well optimized."
Tim Addison, Logistics Data Systems.

"386 protected mode support is fantastic, especially the access to large amounts of memory. It's mainframe compute power on a PC." **Dan Eggleston, Viewlogic.**

"The preprocessor supplied with Professional Pascal is quite useful. The code quality and control over segmentation and memory models are superior to MS Pascal." **Bob Wallace, QuickSoft.**

Check Out These Reviews

- **High C ™:**

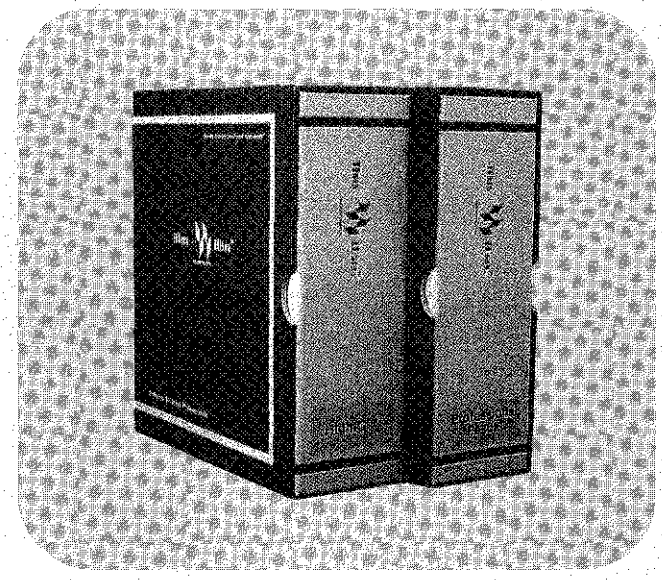
Computer Language	February 1986, '87	
Dr. Dobb's Journal	August 1986	
PC Magazine	Jan. 27, 1987	(80386 version)
Dr. Dobb's Journal	July 1987	(80386 version)
BYTE Magazine	November 1987	(80386 version)

- **Professional Pascal ™:**

PC Magazine	Dec. 29, 1985	
Computer Language	May 1986	
PC Tech Journal	July 1986	
Journal of Pascal, Ada, & Modula-2	Nov.-Dec. 1986	
BYTE Magazine	Dec. '86, June '87	(80386 version)

Why MetaWare compilers

- They are specifically designed for serious software developers.
- They are reliable and robust: they don't break at every turn.
- Their generated code is the best, or near best, on each architecture.
- Their superior diagnostic messages help you produce better products more quickly.
- Your source can be ported with ease to the most popular systems.
- You can link mixed-language modules from our compilers, others
- You can benefit from high-level, personal technical support.
- You can take advantage of the latest ANSI C extensions, and/or extensive Pascal extensions. **High C** has been tracking the ANSI Standard for two years; **Professional Pascal** will soon have a VS Pascal compatibility switch and several Apollo Pascal ext'ns.
- You can take advantage of the latest 387 and Weitek 1167 support — we have the only compilers with Weitek real mode support.



Power Tools for Power Users

Ashton-Tate: dBase III Plus, MultiMate; Autodesk: AUTOCAD, AUTOSKETCH (8087, '387, Weitek); Boeing Computer Services (Sun); CASE Technology (Sun); CAD/CAM giant Daisy ('86, '386, VAX); Deloitte Haskins & Sells; Digital Research: FlexOS; GE; IBM: 4.3/RT, 4680 OS; Lifetree Software (Pascal): Volkswriter Deluxe, GEM-Write; Lugaru: Epsilon; NYU: Ada-Ed cmplr; Semantec: Q&A; Sky Computers; ... (Product names are trademarks of the companies indicated.)

Industrial-Strength

MetaWare C and Pascal compilers are designed for professional software developers. These tools are loaded with options to control them for special purposes. You can adjust the space-time trade-off in code quality. You can adjust external naming conventions to agree with linkers and operating systems. You can specify segment names for segmented architectures, and to help place code or data in particular places for embedded applications. You can select from five memory models for the 8086 family. And on and on.

A Partial List of Optimizations

Common subexpression and dead-code elimination, retention and reuse of register contents, jump-instruction size minimization, tail merging (cross jumping), constant folding, short-circuit evaluation of Boolean expressions, strength reductions, fast procedure calls, automatic mapping of variables to registers (where advantageous), ...

"Platform" — Code Quality

Sun,Apollo,SGI	— 18%, 3%, 26% > resident compiler (Dhrystone).
PC: DOS, OS/2	— 3-10% > Microsoft C; 30% > MS Pascal, LatticeC.
386 32-bit DOS	— no competitors, since November, 1986.
286, 386 UNIX	— 66% better than pcc (Dhrystone, 386).
VAX VMS	— ≈ DEC's excellent C and Pascal; better features.
VAX Ultrix	— 19% > pcc (Dhrystone); much > Berkeley Pascal.
RT PC/4.3bsd	— 89% > the original port of pcc (Dhrystone).
370 CMS,UNIX	— much better than any C, and VS Pascal.
AMD 29000	— >40,000 Dhrystones! Available in Q2, cross.

(408) 429-6382, telex 493-0679. ***Since 1979.***

MetaWare™
INCORPORATED

903 Pacific Avenue, Suite 201 • Santa Cruz, CA 95060-4429

The Clear Choice for Large Programming Projects — PC Tech J.

© 1987 MetaWare Incorporated. MetaWare, High C, Professional Pascal, and DOS Helper are trademarks of MetaWare Incorporated. Others and their owners are duely respected.

Number 45 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal for IBM Micro Programmers

PROGRAMMER'S Journal™

Volume 7.3
May/June 1989

Contents

4 Notes from the Editor

6 Letters
Clark A. Calkins, Keith R. Spitz, Patrick D. Farrell, John O. Goyo

10 PJ Forum: Neural Nets *by Tom Waite and Hal Hardenbergh*
A history of neural nets, breakthroughs, learning, training, and example BASIC code

24 On Graphics: Fast Line Drawing for the EGA and VGA *by Michael Abrash*
Bresenham's algorithm implemented for the EGA/VGA in both C and assembly language

42 Database: Embedded SQL for a Relational Database *by Richard Riehle*
Programming with SQL embedded in COBOL, Pascal, and C

52 Database: Btrieve for C Programmers *by Amrit Khalsa and Pete Maclean*
A clean C interface to the Btrieve file manager

56 Database: Converting Database Files Using Filters *by M. Steven Baker*
Quick-and-dirty file translation for your database needs

61 C: The Proposed ANSI C Language Standard *by Rex Jaeschke*
A look at WATCOM's latest offering and ECO C for ANSI conformance

64 Windows: A Substitute for the Missing TYPE Command in Windows *by William S. Hall*
Font selection, file open dialog boxes, accelerator keys, and multitasking

74 C++: User Interface Tools for DOS *by Vaughn Vernon*
DOS classes for the display, keyboard, and mouse

86 Miscellaneous *by M. Steven Baker*
dBASE C Libraries
CodeBase (Sequiter Software)
Apex Database Library (Apex Software)
Software Tools
Optlib and Optlib/2 (SLR Systems)
Inside! (Paradigm Systems)
ASMFLOW (Quantum Software)
FORTRAN Tools
Prism Toolkit (OTG Systems)
Conferences and Exhibits
Software Development '89
Books
A Guide to the SQL Standard,
by C. J. Date (Addison-Wesley)

95 PJ Billboard

96 Advertiser Index

The database jungle is filled with perils, whether obvious or lurking deep within the heart of darkness. This issue of *PJ* offers tips, tricks, and powerful tools to help you tame the database jungle.

Artist:
Larry Milam

Art Director:
Ralph Bentley

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

SERIOUS DEBUGGING *at a* REASONABLE PRICE

All the speed and power of a hardware-assisted debugger at a software price

Hardware-level break points

REAL-TIME break points on memory locations, memory ranges, execution, I/O ports, hardware and software interrupts. More powerful break points than ANY software-only debugger on the market. Soft-ICE gives you the power of an in-circuit emulator on your desk.

Break out of hung programs

With a keystroke - no external switch necessary. Even with interrupts disabled.

Breaks the 640K barrier

Soft-ICE uses ZERO bytes of memory in the first 1MB of address space. This is especially useful for those subtle bugs that change when the starting address of your code changes. With Soft-ICE your code executes at the same address whether the debugger is loaded or not.

Works with your favorite debugger

Soft-ICE can be used as a stand-alone debugger or it can add its powerful break points to the software debugger you already use. You can continue to use your favorite debugger until you require Soft-ICE. Simply pop up the Soft-ICE window to set powerful real-time break points. When a break point is reached, your debugger will be activated.

Solve tough systems problems too

Soft-ICE is ideal for debugging TSRs, interrupt handlers, self booting programs, DOS loadable device drivers, non-DOS operating systems, and debugging within DOS & BIOS. Soft-ICE is also great for firmware development because Soft-ICE's break points work in ROM.

How Soft-ICE Works

Soft-ICE uses the power of the 80386 to surround your program in a virtual machine. This gives you complete control of the DOS environment, while Soft-ICE runs safely in protected mode. Soft-ICE uses 80386 protected mode features, such as paging, I/O privilege level, and break point registers, to provide real-time hardware-level break points.

"Soft-ICE is a product any MS-DOS developer serious enough to own a 386 machine should have."
Dr. Dobb's Journal — May 1988

RUN CODEVIEW IN ONLY 8K!

CodeView is a great integrated debugger, but it uses over 200K of conventional memory. MagicCV uses advanced features of the 80386 microprocessor to load CodeView and symbols in extended memory. This allows MagicCV to run CodeView using less than 8K of conventional memory on your 80386 PC.

Don't let 640K be your limit!

If you are closing in on the 640K limit and would like the power of CodeView, MagicCV is for you.

Don't let the debugger hide the bug!

Even if you're not closing in on the 640K limit, running CodeView with MagicCV makes your debugging environment much closer to the end user's program environment. You can use CodeView to locate subtle bugs that only occur when there is plenty of free memory, or those difficult bugs that only occur when your program is running with a couple of TSRs loaded.

How MagicCV works

MagicCV uses the 80386 to create a separate virtual machine for CodeView. MagicCV uses between 4K & 8K of conventional memory as a bridge between the DOS environment and CodeView.

MagicCV is easy to use

If you are a CodeView user, you already know how to use MagicCV too. Just type MCV instead of CV; everything else is automatic.

NEW
MS Windows Version

MagicCV with Soft-ICE

Using Soft-ICE with CodeView gives you the features necessary for professional level systems debugging. MagicCV and Soft-ICE can work in concert with CodeView to provide the most powerful debugging platform you will find anywhere.

MagicCV	$199
Soft-ICE	$386

Buy Both and Save $86!

CALL TODAY
(603) 888 - 2386
***or FAX* (603) 888 - 2465**

30 day money-back guarantee
Visa, Master Card and AmEx accepted

NU-MEGA TECHNOLOGIES

P.O. BOX 7607 • NASHUA, NH 03060-7607

Both require 80386 AT compatible or IBM PS/2 Model 80. MagicCV requires at least 384K of extended memory. CodeView is a trademark of Microsoft Corporation.

Number **160** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Resource Journal for IBM Micro Programmers
PROGRAMMER'S Journal™

NOTES from the Editor

Editor
M. Steven Baker
Managing Editor
Robert E. Wanetick
Contributing Editors
Michael Abrash, Frank D. Greco, Tom Swan

Contributing Writers
Michael Abrash
M. Steven Baker
William S. Hall
Hal Hardenbergh
Rex Jaeschke
Amrit Khalsa
Pete Maclean
Richard Riehle
Vaughn Vernon
Tom Waite

Publisher
Elizabeth Oakley
Associate Publisher
Bobbi Sinyard
Production
Ralph Bentley
Scott E. Pringle
Illustration
Jim Carpenter
Advertising
Robert Tappan
Andrea McHone
Subscriptions
Gary Hoyer
Administrative Assistant
Lori Fanslow
Founder
Greg Estes

Business and Editorial Offices
Programmer's Journal
150 North 4th Street
Springfield, OR 97477
503-747-0800

Subscriptions: 1 year (6 issues) $24
($30 in Canada and $40 elsewhere)
Change of Address
Programmer's Journal
P.O. Box 30160
Eugene, OR 97403

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited and published.

Code and listings in *Programmer's Journal* appear "as is," and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy, or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Robert Tappan or Andrea McHone at 503-747-0800, or write to Programmer's Journal, P. O. Box 30160, Eugene, OR 97403.

Programmer's Journal is produced using the following software products: Ventura Publisher™, Aldus PC PageMaker®, and CorelDraw. The PostScript™ files are output on a Linotronic™ 200.

Programmer's Journal (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company. Second-class postage paid at Springfield, OR, and additional mailing offices.

POSTMASTER: Send address changes to Programmer's Journal, P.O. Box 30160, Eugene, OR 97403.

© 1989 OAKLEY PUBLISHING COMPANY

Of the big three computer applications (word processing, spreadsheets, and databases), database programming is most like the jungle in our cover metaphor. By their nature, database systems have usually required custom programming. Most of the PC world has adopted the dBASE III file format and language as a de facto standard. The original dBASE II product from CP/M-80 never had any major competitors, and the early DOS slate started in a similar fashion.

Several worthy competitors have challenged the dominance of dBASE. R:BASE (Microrim) offered convenient data dictionaries and automatic application generation. Paradox (now owned by Borland) provided the powerful query-by-example metaphor and excellent network support. Various products emerged for creating custom database systems—db Vista (Raima) using the network model, Zim (Zanthe) based on the entity-relationship model, a number of B+ tree systems, and file managers like Btrieve (now owned by Novell) and AXOS (McSoft). The PC marketplace also fostered various dBASE emulators, compilers (FoxBase, Clipper, etc.), programming libraries, and translators. Database systems were also ported from the minicomputer/UNIX world—INFO, Oracle, Informix, etc. We have yet to see the Microsoft database product (code-named Omega), but one would be naive not to expect a competitive product.

SQL—Long Live the New King?

Now SQL is to become the new PC database standard bearer. SQL comes in two flavors—interactive and embedded. Interactive SQL is not that different from the dBASE dot prompt. Clearly, most database programming with SQL will eventually be embedded, just as most dBASE applications depend on the dBASE language. Although ANSI published a standard for SQL in 1986, this bears only a limited resemblance to commercial SQL products from IBM, Oracle, and others. The de facto SQL standard lies somewhere between ANSI SQL and the necessary extensions made by IBM and others for a commercial product. We now have various PC SQL servers, DOS implementations of mainframe SQL products, and IBM's OS/2 Extended Edition with the Database Manager now part of the operating system. Are we to witness a "passing of the torch" on the PC?

A lot of inertia is in place with existing products. Consider the vast hordes of skilled programmers and users familiar with existing PC database products. The mainframe world is also experiencing a large reverse migration of software products from the PC. The dBASE language is now available on the Mac, XENIX (FoxBase), UNIX (WordTech), and a VAX/VMS version from Recital. dBASE C library products, Raima's db Vista, and other products also ease PC database migration to other platforms. dBASE language front ends are being grafted onto various foreign SQL servers (WordTech). But attempts to graft SQL on top of dBASE (at least the current dBASE IV release) will need major speed improvements to be a viable product. Ashton-Tate may have been correct in suing Fox Software regarding ownership of the dBASE language rather than over "look and feel" issues. This is certainly a market in flux.

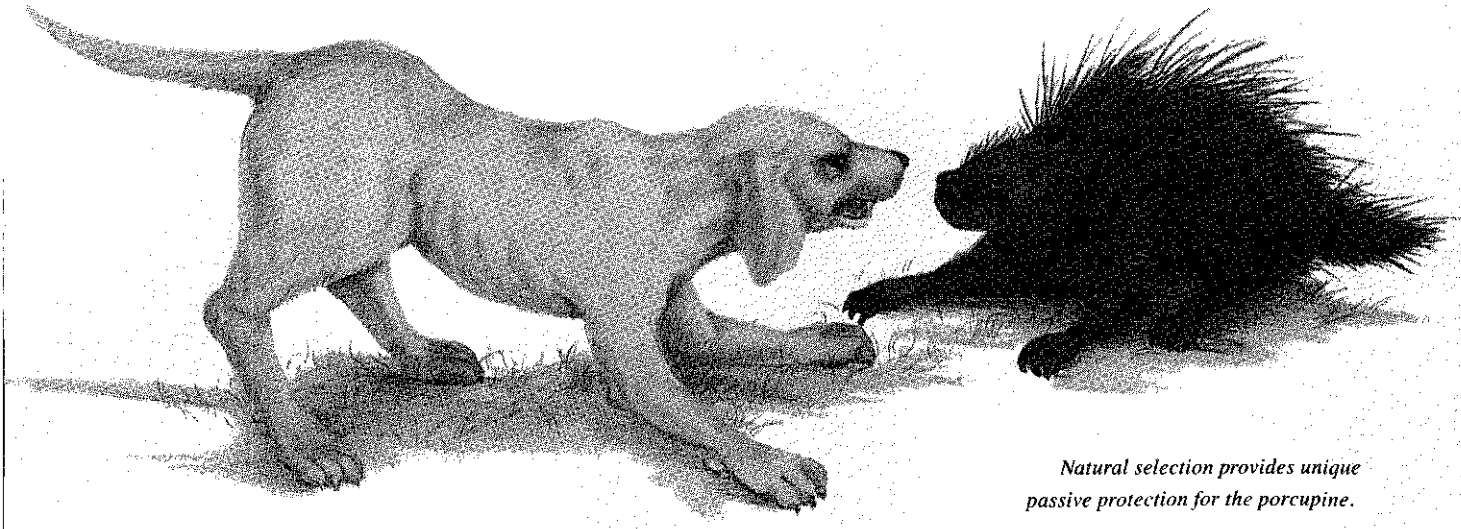
Ah . . . Technology in the Air

As Hal Hardenbergh so eloquently wrote, technology is in the air, and everyone in the industry breathes it. Weitek has announced an Abacus 3168 single-chip math coprocessor for the Motorola 680X0 CPU family. Custom floating-point accelerators have been available on 680X0 workstations (SUN, etc.) for some time. But the 3168 ($500 in volume) will make the Motorola family more competitive with the 386. Meanwhile, Definicon has a SPARC coprocessor board for the PC along with new 25-MHz (and soon 33-MHz) 68030 boards. Yet, buying your overpriced Apple Macintosh MacII gets you a half-speed 16-MHz clock. Yarc sports a 50-MHz AMD29000 RISC coprocessor board for the PC, while Intel has shown its own i860 RISC PC coprocessor board. Soon Compaq, AST, Everex, and (one assumes) IBM will be delivering official 33-MHz 386 boxes (Compaq and AST using EISA). The Intel 80486 will be officially announced, and SUN will have new inexpensive SPARC and 68030 workstations.

Magazines, They Come and Go!

Another one has passed, this time *PC Tech Journal*, from Ziff-Davis. Yet somehow *PJ* has grown and muddled its way through six and a half years. Maybe we're doing something right—or at least not wrong. We hope it's the former. —*Steve*

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Natural selection provides unique passive protection for the porcupine.

The Activator - Natural Selection For Software Protection

Inventor and entrepreneur Dick Erett explains how "The Activator" provides sane protection for your intellectual property.

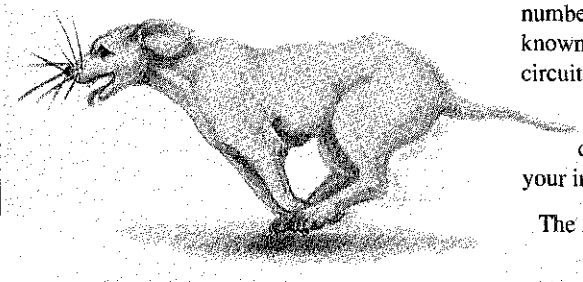
"In any industry, just as in nature, the process of natural selection raises one solution above another. Natural selection is the most elegant of engineers.

In the area of software protection The Block has been selected by the marketplace as the solution that works. Over 500,000 packages are protected by our device.

For the past 4 years our philosophy has been; *'You have the right and obligation to protect your intellectual property.'*

A New Ethic For Software Protection

In allowing end-users unlimited copies of a software package and uninhibited hard disk and LAN operation, The Block has created a new ethic for software protection.



By removing protection from the magnetic media we remove the constraints that have plagued legitimate users.

They simply attach our key to the parallel port and forget it. It is totally transparent, but the software will not run without it.

A New Technology For Software Protection

Our newest model, The Activator, builds on our current patented design, and establishes an unprecedented class of software protection.

We have migrated and enhanced the circuitry of The Block to an ASIC (Application-Specific Integrated Circuit) imbedded in The Activator.

This greatly improves speed and performance, while reducing overall size. Data protection can also be provided.

Programmable Option

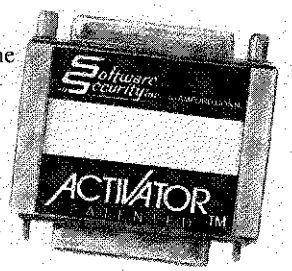
The Activator allows the software **developer** the option to program serial numbers, versions, or other pertinent data known only to the developer, into the circuit, and access it from the program.

Once you program your part of the chip, even we have no way to access your information.

The ASIC makes emulation of the device virtually impossible. It also presents an astronomical number of access combinations.

Full 100% Disclosure

Since The Activator is protected by our patent we fully disclose how it works. Once you understand it, endless methods of protection become evident.



Just as no two snowflakes are the same, no two implementations of The Activator are identical. And like the snowflake the simplicity of The Activator is its greatest beauty.

We never cramp your programming style or ingenuity. Make it as simple or complicated as you desire.

Let us help safeguard what's rightfully yours. Please call today for additional information or a demo unit. *It's only natural to protect your software.*"

1-800-333-0407 ext.103
In Connecticut 203-329-8870
Fax 203-329-7428

Software Security inc.
1011 High Ridge Road
Stamford, CT 06905

Unlimited Copies • Programmable • No Batteries • Small Size • Fast • Patented • Data Protection

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Confused Readers . . . ?
My compliments to you and *Programmer's Journal* on the mention of several of the fine disassemblers on the market today. This subject has stayed in the closet long enough. I realize that this was not intended to be a comprehensive review, but I would like to clear up a couple of points where your readers may have been confused. MD86 label names are remembered in uppercase and lowercase characters just as they are entered (and in this way they are "case sensitive," I suppose). However, all searches of the label table are case insensitive; thus, the user does not have to be concerned about the case of the labels.

The table limits mentioned, while they do exist, should be thought of in context. The assembler used (MD86 assumes this will be MASM V4 or later) also has limits that, in many cases, are more restrictive. Also, limits are being relaxed as newer versions are released. The only 80X86 instructions I am aware of that MD86 has any "trouble" with are those that MASM itself cannot assemble because of its syntax requirements.

Hats off to your cartoonist! I cannot read the signature (intentional?), but it sure livens up your magazine. Keep it up.

Clark A. Calkins
C.C. Software
Walnut Creek, California

[Jim Carpenter is our modest illustrator. His name appears in the masthead. —REW]

More DOS Tips
Ahhh, the intricacies of MS-DOS! Raymond Michels' "Undocumented DOS Internals" article was very enlightening. I'd like to try to return at least part of the favor. The information after the double null byte in the environment is described briefly in the IBM DOS 3.3 *Technical Reference Manual:*

> Following the byte of zeros that terminates the set of environment strings is a set of initial arguments passed to a program that contains a word count followed by an ASCIIZ string. The ASCIIZ string contains the drive, path, and filename{.ext} of the executable program. Programs may use this area to determine where the program was loaded from.

It doesn't exactly define the function of the "word count" (do take note, however, that it is a **word** value and not a **byte** value), but since it is always "1" (followed by one ASCIIZ string), it seems likely that its purpose is to tell how many ASCIIZ arguments follow.

Anyway, thanks again for the tips. Keep 'em coming.

Keith R. Spitz
Bellevue, Washington

Documented Opcode 83H
I am surprised by a statement made by Mr. Larry Smith in his letter in the November/December issue of *PJ*, Volume 6.6.

Mr. Smith claims that Microsoft's Macro Assembler Version 5.1 makes use of Intel 80X86 opcode 83H as an undocumented opcode, citing as his authorities "Intel documentation and Russell and Alexy's Blue Bible" and the AFD debugger's lack of recognition of the 83H opcode. No mention is made of the documentation that normally accompanies the assembler that he is criticizing.

Were he to consult Microsoft's spiral-bound *Macro Assembler Reference*, 5.0 or 5.1, he would find that, not only is opcode 83H documented, 83H is the most documented opcode in the book! The 83H instruction formats not only are given on pages 37, 38, and 39 but also provide the encoding interpretation example given on page 29.

Perhaps the 83H opcode isn't shown in whatever documentation Mr. Smith possesses—or, more to the point, does *not* possess. 83H certainly is thoroughly documented with

NEW VERSION 4.0
DYNAMITE DEMOS
Proteus Prototype/Demo System

Why fizzle with an outdated demo program when you can explode with Proteus?

Proteus is the first prototype/demo system to combine complete simulation capability, a powerful Screen Painter, video effects, custom sound and much more in a program you don't have to be a genius to figure out.

And now Proteus supports all graphic modes for the CGA, EGA and VGA in the popular PCX file format.

"A joy to use."
. . . Computer Language

■ Complete Simulation Capability
■ Powerful Screen Painter
■ Bit-Mapped Graphics
■ Video Effects
■ Custom Sound

Don't Let Your Next Demo be a Dud!
Find out why software developers from Ashton-Tate to ZyLogic have switched to Proteus.

1-800-634-9986

$149. No royalties. 30-day Money Back Guarantee. MC/VISA/COD/PO accepted.

Helios
P.O. BOX 22869, SEATTLE, WA 98122

Number 258 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PERSONAL REXX 2.0

NEW PERSONAL PROGRAMMING POWER

Personal REXX is a powerful general-purpose language bringing mainframe command and macro language capabilities to the PC. Personal REXX 2.0 is a new, faster version that runs larger programs, and features:

- **A superset of IBM CMS REXX**
- **The ability to develop powerful batch procedures**
- **Compatibility with IBM SAA REXX command language**
- **Windowing and menu support**
- **Easy prototyping**
- **The ability to interface with KEDIT™ text editor and HyperTerm™ communication package**

Personal REXX runs on the IBM PC, AT, PS/2, and true compatibles using DOS 2.0 or later. A Programmer's Journal review said: ***"Everyone who tries it seems to love it."*** Call to find out why, and how you can enjoy the programming power and the many benefits of Personal REXX 2.0

Personal REXX Version 2.0 is available at $150. Add $3 shipping. MC, VISA, American Express.

MANSFIELD
Software Group

P.O. Box 532, Storrs, CT 06268
(203) 429-8402

KEDIT is a trademark of the Mansfield Software Group, Inc.; HyperTerm is a trademark of Quercus Systems; IBM and IBM PC, AT and PS/2 are trademarks of International Business Machines.

Number **233** on the Reader Service Card

People are talking about us.

When professional FORTRAN programmers develop or port large programs they use Lahey's F77L-EM/32 and F77L-EM/16, *PC Magazine's 1988 Technical Excellence Award Winners.* F77L-EM/32 is a fast 32-bit protected-mode compiler that accesses up to 4 gigabytes of memory on 80386s. F77L-EM/16 gives 80286 users the power to create 15 megabyte programs. These protected-mode FORTRANS include the features that have made them, and our F77L and Lahey Personal FORTRAN, market leaders: full ANSI 77 Standard, VAX and IBM VS extensions, fast compilation, comprehensive diagnostics, and a powerful debugger.

TECHNICAL EXCELLENCE
PC MAGAZINE
WINNER
COMPILERS/LANGUAGES
F77L-EM/16 AND F77L-EM/32
LAHEY COMPUTER SYSTEMS, INC.
1988

Contact us to discuss our products and your needs. (800) 548-4778
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123 Tlx: 9102401256

Number **44** on the Reader Service Card

RADs™

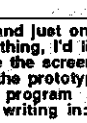
works the way you think.

Rapid Application Development System

We take you from Concept to Prototype to Finished Program to Demo to Tutorial

Read our little scenario

"So, Your	• Boss • Client • Market
says: "I need a:	• Program • Prototype • Demo • Tutorial • Presentation • All The Above
"with:	• Nested Menus • Pop-up windows • Pull-down Menus • Light Bars • Original User Interface • Music
"Oh, and in the prototype, demo, and tutorial could you make it:	• Interactive • Self-Running • Both
"And it needs:	• User Input • Simulated Report Generation • Active Screen Areas for User Input • Dialogue Boxes • Variable Math Calculation • GO SUB Routines and much more
"Oh, and one more thing, I need to:	• Make as many copies as I want with no license fees
"And:	• No screen in front promoting the product it was developed in.
"And:	• Free 800 # telephone support
"Oh, and just one more thing, I'd like to use the screens from the prototype in the program we're writing in:	• C • Basic • Pascal • dBase • Clipper
"And the routines should also do:	• Screen Handling • File Handling • Mouse Support • Serial Port Control



"And could I have that in 2 days please?" Sound familiar?

NO Problem, if you've got ***RADs !***

G·E·N·E·S·I·S

GENESIS DATA SYSTEMS
8415 Washington Place N.E.
Albuquerque, NM 87113 (505) 821-9425

◆

COMPLETE SYSTEM: $299.95
Demo Available

To Order or For Product Information Call

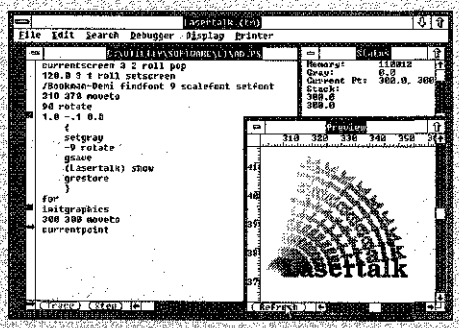
1-800-777-1437

Number **274** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Master PostScript®

Language Programming - On Your PC



Lasertalk™ PC

A PostScript language development environment which provides interactive access to the PostScript interpreter, a PostScript language debugger/editor, on-screen previewer, dictionary browser and Adobe Systems' operator descriptions, online. Requires PC/AT or greater and a PostScript printer. Includes Microsoft® Windows runtime. Suggested Retail Price $399. For more information, see your dealer or contact Emerald City Software at (415) 324-8080.

EMERALD CITY SOFTWARE

800 Menlo Ave. Suite 102, Menlo Park, CA 94025
(415) 324-8080

Lasertalk is a trademark of Emerald City Software. PostScript is a registered trademark of Adobe Systems, Inc.
Microsoft is a trademark of Microsoft Corporation.

Number **241** on the Reader Service Card

MASM 5.0 and 5.1 that I purchased from Microsoft.

Patrick D. Farrell
Shoreview, Minnesota

PJ Bound?

I am going to make a bold request.

While I would like to obtain a number o *PJ* back issues, it would cost me over $100 to get all those of interest to me, which is too dear a price.

On the other hand, I would gladly (and eagerly) pay $40 for a "bound volume" (in the style of *DDJ*) of the first five years. How feasible would that be? Perhaps you could survey your readership to see how many people would purchase such an object.

If I lived within easy commuting distance I would even volunteer to collate the back issues and then mask out the advertisements.

John O. Goyo
Port Credit, Ontario

[Thanks for the offer. We are considering publishing PJ *anthologies and bound volumes. Readers, any special requests? Any other volunteers? —REW]*

Programmer's Journal likes to hear from its readers. Please address correspondence to Managing Editor, Programmer's Journal, 150 N. 4th St., Springfield, OR 97477. You can also contact us via fax (503-746-0339) or MCI Mail (user name OAKLEY).

PJ CODE

Programmer's Journal Code DiskScription—

Typing in long code listings uses up a lot of valuable time, and calling up bulletin boards doesn't appeal to everyone. So what does the modern programmer do? Get a ***PJ Code DiskScription***, of course!

What's a PJ Code DiskScription?
All the code listings from six consecutive issues of *Programmer's Journal*, sent directly to you every other month on disk, for the money-saving subscription price of only $39.95.*

You'll get all the upcoming installments of Michael Abrash's EGA/VGA ASM code. You'll get more of Tom Swan's handy Pascal code. You'll get great code from all ***PJ***'s authors, some of the country's hottest programmers.

You'll receive your code disks in the mail every other month throughout the year. Learn from ***PJ*** code or use in it your own programs. Each disk contains all of the program listings published in ***PJ***plus .EXE files, .OBJ files, benchmark source files, all for **only $39.95!***

If purchased separately, the regular price for these six disks is $60.00. **So save yourself some time, effort and over $20.00—order your *PJ DiskScription* today!**

GUARANTEE: If you are dissatisfied with your PJ Code DiskScription for any reason, we will gladly send you a full refund on all unmailed diskettes.

Code disks from six issues, only

$39.95*

PJ CODE DELIVERED DIRECTLY TO YOU—
GET YOUR *PJ Code DiskScription* TODAY

All *PJ Code DiskScriptions* must be prepaid.
Return the card on the opposite page with your payment.
For faster service with VISA MC call toll-free: 800-234-0386

* **Prepaid, US only.** Call or write for prices outside the US.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

HARDCORE SOFTWARE

Stop fooling around. It's time to get hardcore about software. With Microsoft.

We'll give you all the resources you want. Tens of millions in R&D funding. Along with one of the most elementary tools for thinking — a door, which leads to your own private office. All backed by management that truly *does* speak your language, because they probably helped write it.

We're serious about software design. If you are too, then apply right now for one of these opportunities.

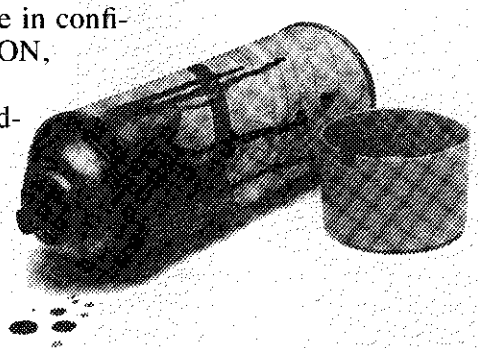
Software Design Engineers

We're working on everything from worldclass application software, compilers, operating systems, and networking to sophisticated graphics, powerful productivity software and more. You could be too, if you have programming experience and a background that includes 68000, Macintosh® Toolbox, Windows™, "C", micro's, 8086, UNIX™/XENIX®, or MS-DOS®.

There are opportunities to work with our teams in systems, applications, CD-ROM, languages, or networking.

If you qualify, Microsoft offers you an opportunity to live and work where the quality of life is high and the cost of living is low — the beautiful Pacific Northwest. Along with amenities such as a health club membership, workout facilities and parcourse, plus an array of benefits.

To apply, please send your resume in confidence to: MICROSOFT CORPORATION, Recruiting, Dept. CWSDE-0589PJ, 16011 N.E. 36th Way, Box 97017, Redmond, WA 98073-9717. We are an equal opportunity employer. No phone calls, please.



Microsoft®

Trademarks are registered to their respective companies.

Number 232 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Neural Nets

by
Tom Waite and Hal Hardenbergh

This article is not about artificial human beings. It's not even about artificial human brains. It is about neural nets or, more exactly, artificial neural net models.

There are some pretty exciting things going on in the field of neural nets these days. Massive parallelism works: Adding *n* devices provides an *n*-times performance increase even when *n* is very large. Neural nets can learn. And real-world applications for neural nets already exist!

For these and other reasons, neural nets are suddenly a hot area of research and development. The field is obviously brand new. In fact, it is so new that when we were researching this article, the oldest popularization we had in one of our homes was in the March 1975 issue of *Analog*, the science fiction magazine!

Hmmmm . . . Brand new? . . . Popularization in 1975? . . . This requires an explanation.

A guy named Rosenblatt had been doing research in the field of natural neural nets for years. In 1958 he described the "perceptron," an artificial electronic neural net–like device. The perceptron generated a lot of excitement in the research community. It suggested that a paradigm of artificial intelligence (AI) very different from the then current (and still current) LISP–expert system paradigm based on sequential von Neumann processors could be developed. (The 1975 *Analog* article was about perceptrons.)

In 1960, Widrow and Hoff introduced the adaptive linear filter, emphasizing mathematical rigor in the learning (or training) process. Research on the perceptron benefited from the Widrow–Hoff learning rules, and the adaptive linear filter was a success. The research community loves mathematical rigor, and artificial neural nets became a powerful competitor of conventional AI, which is based on expert systems and LISP.

You Vill LISP

Research dollars that were spent on artificial neural nets were not spent on LISP. Enter Marvin Minsky and Seymour Papert, who both were heavily involved in the LISP paradigm of AI. It turned out that the perceptron had a few "gotchas"—there were some problems that could not be solved by the perceptron. Because of these problems, the perceptron was buried. The principal gravediggers were Minsky and Papert, whose 1969 book *Perceptrons* identified every wart and imperfection of the perceptron.

Their book is widely regarded as a magnificent analysis of the perceptron. The last chapter had a small problem, however: In it, Minsky and Papert essentially stated not only that the perceptron did not work, but that nothing along those lines would ever work! (They did not say, "Everybody get back to work on LISP-based expert systems," but some folks inferred that.) Research—and, more important, funding for research—dried up. The AI community turned almost exclusively to expert systems based on serial processors.

Lots of senior scientists have made mistakes when predicting that something "will never work." One prominent physicist once asserted that "nuclear power plants are impossible, because nuclear fission cannot be controlled"! Hint: The physicist's first name was Albert.

It had been known from the beginning that certain types of problems might require multiple-neuron and multiple-layer versions of perceptrons or adaptive filters. After all, the researchers were all working on artificial neural net models, and natural neural nets (brains) do have multiple neurons and multiple layers. The problem was, nobody knew how to "train" such a complex device. The math did not exist, so Minsky and Papert were, for about 15 years, correct.

But in 1981 Rumelhart and McClelland got together (they were both at U.C.–San Diego) and proposed a multiple-layer network for word and character recognition. This early (23 years after the perceptron!) model of what are now known as neural nets used preassigned weights, not learning. But the PDP (Parallel Distributed Processing) group coalesced around these two, attracting researchers such as Francis Crick, of DNA fame.

Thus, research in artificial neural nets was revived. In 1985 one solution to the mathematical problem of how a multi-element and multilayer net could learn appeared. Minsky and Papert's assertion that neural nets (perceptron-like devices) could never solve certain problems was proved wrong. In 1986 Rumelhart, Hinton, and Williams described an improved learning algorithm that was a generalization of the 1960 Widrow–Hoff method. This algorithm is the working foundation of today's neural nets. It turned out that the algorithm had been developed (in a different context) in 1974!

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

It's not unusual for "new" mathematical techniques to prove not so new. Cooley and Tukey described the Fast Fourier Transform, or FFT, in 1965. The FFT is based on the reasonable principle that one should never perform the same mathematical operations on identical data more than once. It turns out that electrical engineers were using that method, called "decimation in frequency," to design array antennas before World War II. Carl Friedrich Gauss used the same method to speed up calculations of orbital elements nearly 200 years ago!

Brainy Power

The human brain uses chemical reactions, with "cycle times" of milliseconds. And yet it has enormously greater processing power than the fastest existing computers, which have cycle times in nanoseconds (ns)—a million times faster. Direct comparisons are obviously difficult, but it is generally believed that the human brain has six to eight orders of magnitude more processing power than existing supercomputers. Most of that processing power is devoted to the visual system and, to a lesser extent, speech. The human brain uses *massive* parallelism to achieve its computing power.

How massive? The human brain is seven orders of magnitude (ten raised to the seventh power) faster than, yet has a cycle time a million times slower than, Cray's Y-MP (ballpark numbers, obviously). If the Cray has a 64-bit CPU, then we can say the human brain has a 640-terabit CPU. Now, that's massively parallel! Too bad the most common use of all that computing power is watching "Gilligan's Island" reruns.

The human brain has $>10^{10}$ neurons and $>10^{14}$ synapses and so does not have enough neurons to form a 640-terabit CPU. What that means is that the human brain is organized more efficiently than a von Neumann computer. Are you surprised?

Fuzzy Classifications

Neural nets are modeled on what is known of the human brain. One thing the human brain does, and a Cray doesn't, is make decisions based on "fuzzy" data. Human babies are superb at making decisions based on fuzzy data—recognizing Mama, for instance. Since Mama wears different clothes on different days, the best way to recognize Mama is to recognize her facial features—whether seen from the left, head-on, or from the right; near or far; smiling or frowning; lipstick and eyeshadow or no.

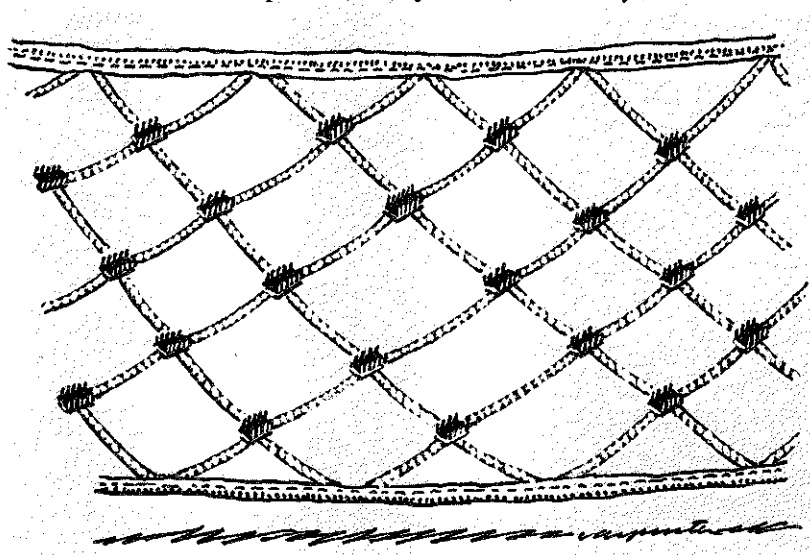
Babies learn to do this. Neural nets are very good at making decisions based on fuzzy data, but not yet so good as a baby. One commercial firm is already selling neural net systems that can recognize a human face. At all perspectives and distances? Well . . . as we said, neural nets aren't so good as a baby . . . yet.

Neural nets are commonly used as classifiers, which is most often a yes/no decision. The Pentagon might like to classify images transmitted from spy satellites: *"Does this image contain an airfield (oil pipeline, missile pad, silo)? **y/n**"* A banker might want to know *"Should I make a loan to this head of the household? **y/n**"* Please don't laugh—one of the most successful applications of neural nets is to augment or supplant a bank's loan officer! The Pentagon's applications are not yet (so far as we know) operational.

Massive Parallelism

If you want to build a really fast artificial image-recognition system, you are going to have to use the same technique the human brain uses: massive parallelism. We are talking *square yards* of silicon.

Talk about funny coincidences: Glasstech Solar is in Wheatridge, Colorado. Glasstech makes sputtering machines that are used by solar cell manufacturers to produce amorphous (not crystalline) silicon by the acre. Glasstech



PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

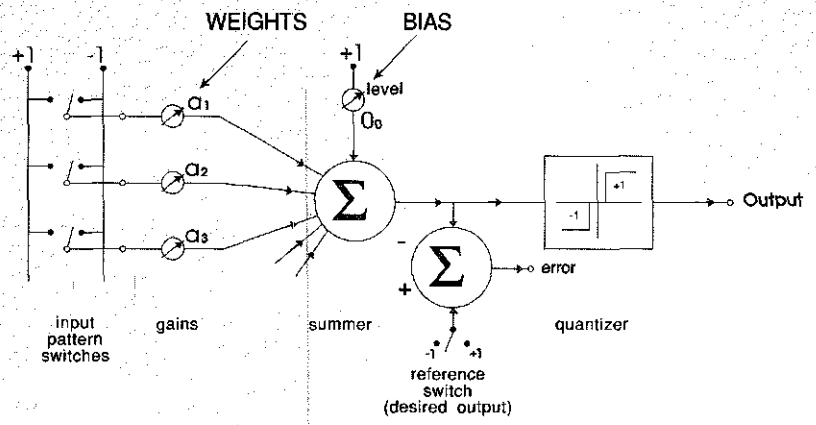


Figure 1

has just shipped a sputtering machine to JPL, in Pasadena, for use in making very large—make that very, very large—arrays of programmable thin-film resistor networks for analog neural nets. Let's see . . . isn't JPL involved in reception of telemetered data from satellites and such?

Adaptive Filters

Widrow and Hoff's Adaline (adaptive linear classifier, 1960) is both son and father to neural nets. It followed, and was obviously influenced by, the perceptron. But the LMS (least mean squares) algorithm proved central to the later development of practical multi-element and multilayer perceptrons.

Figure 1 shows a typical adaptive classifier. Adaline's nonlinear element was simply a hard limiter, with an output of +1 or -1. Each input is assigned an adjustable weight, and all of the weighted inputs are summed, with a bias. The output is compared with the desired output and an error is generated. The weights are adjusted so that the classifier will correctly classify the inputs by minimizing the error for all of the various inputs.

The weighting can be done using analog or digital methods, and there is a lot of current controversy over which method should be used, especially for neural nets. If digital methods are used, the weights are commonly floating-point numbers. Each input is digitized, converted to floating point, and multiplied by its weight, which is a floating-point constant. All of the weighted inputs are summed, and the sum is applied to the nonlinear element. There is an obvious application for a floating-point multiplier accumulator (MAC) here; Weitek makes them. Then we need analog–digital converters on the input, a set of registers or a fast memory bank for the weighting constants and the bias, . . .

It was soon discovered that adaptive linear classifiers made dandy adaptive filters. What's the difference? Filters work on signals over time, usually in real time, where the input is not known in advance. Also, adaptive filters don't use a nonlinear element at the output.

Adaptive classifiers and filters have been around for a long time. They work very well. They have useful real-world applications—modems, for instance—but, like the perceptron, they have fundamental limitations.

Neural Nets

Remember that neural nets are brand new. But it was not possible to train a multi-element and multilayer net until the requisite math was developed in 1985–86. The field did not begin to gather momentum until 1987. It should not surprise you to discover that various researchers and implementers use varying net topologies. The following discussion applies to the most general form of neural nets, which is the multilayer perceptron.

As Figure 2 illustrates, a neural net is like a collection of adaptive filters with additional layers and neurons added. There is the input layer, an intermediate hidden layer, and the output layer. Each input goes to each summer (node). Each path from a particular input to a particular summer is analogous to a synapse; each summer is analogous to a neuron. The nonlinear element following the summer is analogous to how an actual neuron works. But a neural net is not a direct simulation of brain cells; the action of brain cells—what is currently known of that action—is much too complex for us to use as a model.

A neural net is obviously more complex than an adaptive filter. How much more complex depends on the application. In the real-world commercial application of consumer-loan processing, there aren't a lot of inputs and probably not a lot of summers. (We are describing a proprietary commercial product, remember.) Optical character recognition, for instance, can be more complicated. If you digitize a 64x64 input matrix, that's 4096 inputs. If you want to recognize a 60-character set, that's a little over 60 summers, and each input is assigned a weight for each summer—about 250,000 weights! Once we get past the input layer, there are a lot fewer signal paths; so most of the work is in the input layer.

Double-precision floating-point (FP) multiplier accumulators that can cycle in 100 ns have been available since spring 1988. (Faster MACs have recently been introduced.) A single MAC could simulate the 250,000 input-layer weights in 25 milliseconds! If you had Pentagon money to play with, you could get a pipelined throughput of 410 microseconds per character using 60+ MACs. Parallelism works for neural nets even though it (usually) does not work for von Neumann computers.

Gradient-Descent Learning

The trick, of course, is how to determine the weights and the bias. Widrow–Hoff's method, called *gradient-descent learning*, is similar in concept to the familiar Newton–Raphson method. Rumelhart, Hinton, and Williams' generalization of the Widrow–Hoff equations is generally called *back propagation*. Some folks relate gradient descent to a ball

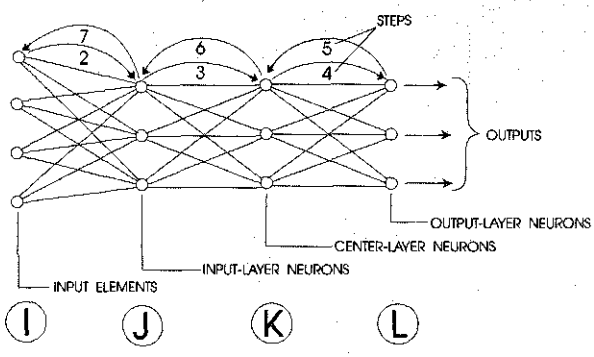
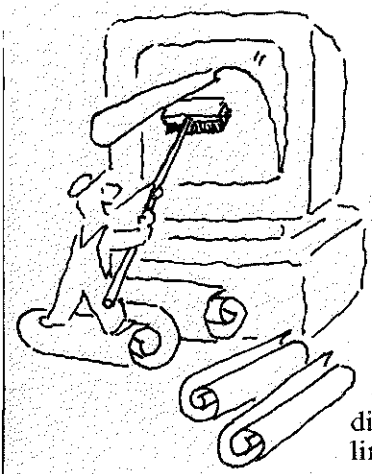


Figure 2

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Two great tools.

SAYWHAT?!
The lightning-fast screen generator.



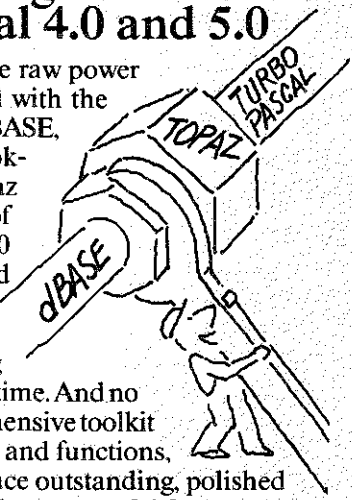
Whether you're a novice programmer longing for simplicity, or a seasoned pro searching for higher productivity, you owe it to yourself to check out Saywhat. You see, with Saywhat, you can build beautiful, elaborate, color-coded screens in minutes! That's right. Truly *fantastic* screens for menus, data entry, data display, and help-panels that can be displayed with as little as one line of code in *any* language.

Here's what you get:

- Design screens, windows, and *moving bar menus!*
- Easy-to-use, powerful editor lets you create screens in a jiffy.
- Pop up your screens and menus with one line of code in dBASE, all the dBASE compilers, your favorite BASIC, Pascal, *or any other language!*
- Screen Library Manager.
- Generates runtime code.
- No runtime license or royalty fees.
- Comes with a 100 page manual, plus dozens of sample programs and free utilities.

$49.95

TOPAZ
The breakthrough toolkit for Turbo Pascal 4.0 and 5.0



If you'd like to combine the raw power and speed of Turbo Pascal with the simplicity and elegance of dBASE, Topaz is just what you're looking for. That's because Topaz (our brand new collection of units for Turbo Pascal 4.0 and 5.0) was specially created to let you enjoy the best of *both* worlds. The result? You can create truly dazzling applications in a very short time. And no wonder. Topaz is a comprehensive toolkit of dBASE-like commands and functions, designed to help you produce outstanding, polished programs, fast. Check out these powerful features:

- Over 200 routines all with easy-to-use, dBASE-like syntax.
- Data entry routines like **SAY, GET, PICTURE, RANGE,** color selection, unlimited data validation.
- Open up to 10 DBF files, with up to 7 indexes each with database routines like **USE, SELECT, SKIP, APPEND, PACK, INDEX ON, SET INDEX TO,** and **FIND.**
- No need to buy dBASE. **CREATE, BROWSE** and **REPORT** utilities included.
- Easily implement Saywhat and Lotus-style moving bar menus.
- **BROWSE** any DBF file with just one line of code! Programmable and windowed too.
- Comprehensive Time & Date math in 7 international formats.
- Powerful *code* and *REPORT generators* included!
- Comes with a complete 250 page manual, plus sample programs to get you started.

$74.95

ORDER NOW.
YOU RISK NOTHING.

Thousands of satisfied customers have already ordered from us. Why not visit your dealer or call toll-free, right now and put Saywhat and Topaz to the test yourself? They're fully guaranteed. You don't risk a penny.

Special limited-time offer! Save $26. Buy Saywhat?! and Topaz together for just **$99** (plus $5 shipping & handling).

Visit your nearest dealer
or call toll-free:
800-468-9273
In California: 800-231-7849
International: 415-571-5019

The Research Group
Software Science, Inc.
100 Valley Drive, Brisbane, CA 94005

MONEY BACK GUARANTEE.
If you aren't completely delighted with Saywhat or Topaz, for any reason, return them within 30 days for a prompt, friendly refund.

Dealers: SAYWHAT?! and TOPAZ are available from Kenfil Distribution, and in Europe from ComFood Software, W. Germany 49-2534-7093

Guaranteed!

THE RESEARCH GROUP

Number 198 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

settling at the bottom of a dent in a surface. The problem is that with *n* inputs, the "surface" has *n* dimensions. The solution involves spatial correlations of partial differential equations.

By now, it's probably occurred to you that the math involved in the training process is a lot more intensive than the very straightforward math used in operation after the weights have been assigned. I don't think you'd want to train our hypothetical character recognizer using an 8088! Still, it's a relatively simple net.

How to Train a Neural Net

So much for the introduction. We are now going to concentrate on training a three-layer perceptron (henceforth the "neural net"). That means we have to examine the algorithmic details of back propagation.

Some terminology: The word *input* is troublesome because it is used in several senses. In an image-related neural net an input is an image, which consists of many pixels, or elements. Not all neural nets are image-related, but almost all have multiple input elements. We will refer to the image as the *input set*, that is, the set of all of the elements of a particular input. The input is in this sense a vector quantity. The input-layer neurons are the neurons that are connected to the input elements.

The input and center layers are commonly called "hidden" layers (only the output layer can be "seen"). We will refer to the "input" and "center" layers rather than "hidden layer 1" and "hidden layer 2." Once a neural net is trained, the process of performing an operation (usually a classification) on an input data set or array (the 1024 values of a 32x32 retina, for instance) is called *recall*.

A neural net is trained by applying an input data set whose desired output is known. Such a data set is called an *exemplar*. We will call the known outputs of the exemplar the *targets*. If we are training a neural net to recognize the uppercase alphabet, we would need at least 26 exemplars. *Learning* is the term applied to the back propagation of the error obtained by comparing the neural net's outputs, computed by recalling an exemplar, with the targets. Learning is accomplished by the back-propagation method.

Training is the process of recalling an exemplar and then learning from that exemplar's output via back propagation. This step is repeated for all exemplars. All of the exemplars taken together are called the *training set*. When the neural net has been trained once using all of the exemplars, we say that a *training iteration* has been completed. The purpose of training is to assign the values of the weights and biases. It commonly requires hundreds of training iterations to train a neural net.

A neural net has inputs and three layers of neurons (see Figure 2). Each input is connected to every input-layer neuron. Each input-layer neuron is connected to each center-layer neuron, and each center-layer neuron is connected to each output-layer neuron. Each neuron has inputs, a bias, and an output nonlinearity. Each connection has an associated weight. The bias can be considered an input weight that is always multiplied by the input value 1. The *sigmoid* is the most commonly used nonlinear function. Its formula is

$$sigmoid(x) = 1 / (1 + e^{-x})$$

Desktop 9-Track Tape Subsystem

Now, 9-track tape lets your micro exchange data with minis and mainframes.

Simply exchange data files on a reel of 9-track tape.

9-TRACK is the first choice for file interchange among data processing professionals. Now, Qualstar's low cost 1/2-inch 9-track *Minitreamer* tape systems bring full ANSI data interchange to IBM PCs or Macintosh, giving your micro the freedom to exchange data files with nearly any mainframe or minicomputer in the world.

Available in both 7" and 10-1/2" versions, compact Qualstar tape drives can sit on your desktop, *using less space than an ordinary sheet of paper*. Systems include DOS or XENIX compatible software, coupler card and cables. High reliability 1600 or 6250 BPI capability may be used for disk backup as well as data interchange. Discover the big advantage 9-track tape has over other micro/mainframe links.

DEC HP ALTOS NCR — IBM MAINFRAMES — AT&T UNISYS APOLLO SUN

Call us today!

#1 Selling 9-Track Systems on the Desktop

QUALSTAR®

FOR DETAILS AND TO ORDER:
FAX (818) 882-4081
PHONE (818) 882-5822
9621 Irondale Ave., Chatsworth, CA 91311

© 1988 Qualstar Corp. All product and company names and trademarks are the exclusive property of their respective owners.

Number 97 on the Reader Service Card

NEW RELEASE!

Build with an Inside! advantage...

Thousands of C, BASIC, FORTRAN and Modula-2 programmers are already using Inside! to improve application performance. You have the best compiler and debugger, now add the performance analysis tool that helps create world class applications!

Inside! analyzes your program in real-time, timing each instance of a function, source line or arbitrary code fragment, all with microsecond accuracy and without source code modification. Utilize the following modes to know where to focus your optimization efforts for maximum reward.

- Function/procedure timing
- Call target timing
- Source Line timing
- Arbitrary event timing
- Non-executed functions or source lines
- MS-DOS function timing
- Overlay and assembly language support

Inside! is available for popular compilers . . .

Call or write today for a free brochure and the latest list of Inside! products.

To Place Orders
(800)537-5043
Product Support
(508)478-0499
Visa/Mastercard/C.O.D. Accepted

$125.00 EACH
Satisfaction guaranteed

PARADIGM SYSTEMS INCORPORATED

Paradigm Systems
P.O. Box 152 Milford, Massachusetts 01757

Inside! is a trademark of Paradigm Systems.

Number 195 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Recall Computations

Let

```
I = the number of input elements; i = array index
J = the number of input-layer neurons; j = array
    index
K = the number of center-layer neurons; k = array
    index
L = the number of output-layer neurons; l = array
    index
```

During recall, the following data arrays are required:

```
Input(i) = the input elements
Iwt(i,j) = the input-layer weights
Ibias(j) = the input-layer biases
Cwt(j,k) = the center-layer weights
Cbias(k) = the center-layer biases
Owt(k,l) = the output-layer weights
Obias(l) = the output-layer biases
```

Of the above data arrays, the input array is the unknown data to be processed ("recalled") by the neural net, and all the other arrays are precomputed values obtained via training. The weight arrays, especially Iwt(*i,j*), can be quite large. Assuming we are using a conventional computer to simulate a neural net rather than hardware for each neuron, we need three more arrays:

```
Iout(j) = the input-layer neuron outputs
Cout(k) = the center-layer neuron outputs
Oout(l) = the output-layer neuron outputs
```

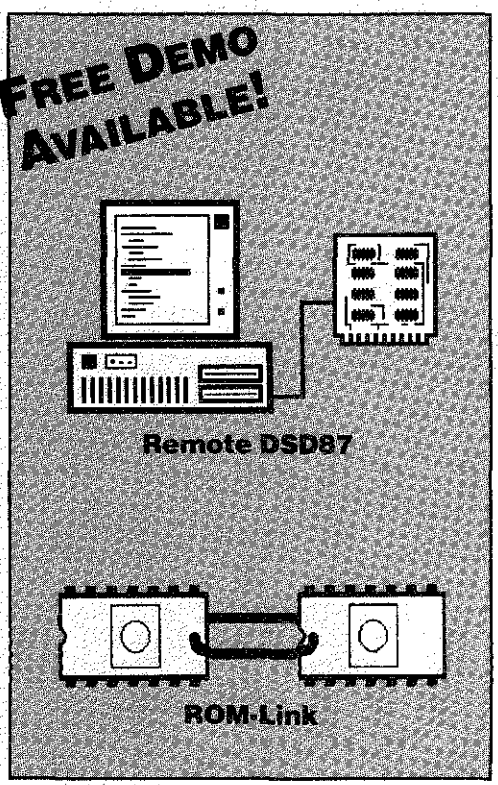
The recall computations involve five steps. First, if the precalculated weights and biases are not stored in EPROM, they must be loaded into the neural net's local memory. This step is performed once, to set up the neural net for operation. The following steps must be performed for *each* recall:

1. Load the input elements into local memory.
2. Perform the sum of products (*convolution*) of the inputs into the first input-layer neuron. Add the bias for that neuron. Calculate the sigmoid function of the resulting value, and assign the result as the output of that neuron. Repeat for all of the other input-layer neurons.
3. Perform the convolution of the input-layer outputs into the first center-layer neuron. Add the bias for that neuron. Calculate the sigmoid function of the resulting value, and assign the result as the output of that neuron. Repeat for all of the other center-layer neurons.
4. Perform the convolution of the center-layer outputs into the first output-layer neuron. Add the bias for that neuron. Calculate the sigmoid function of the resulting value, and assign the result as the output of that neuron. Repeat for all of the other center-layer neurons.

As you can see, steps 2 through 4 are very similar. In step 2, each element of the input array is applied to each input-layer neuron. There are *I***J* paths from particular input elements to particular input-layer neurons. Each path is called a *connection*. There is a weight associated with each connection. During recall, the output for a given input-layer neuron is calculated as follows:

Each input element is multiplied by the weight associated with the connection of that input element to that input-layer neuron. There are *I* such multiplications. The products are summed, which means *I* additions are performed. (This sum-

For the Ultimate in Embedded Systems Programming.



Debug Debug Debug Debug Debug

Remote DSD87 is the best source debugger for embedded development and the only product offering high performance and advanced software In Circuit Emulator capabilities. This package and your compiler form a complete solution for the development of embedded programs. Order now and receive a FREE copy of Native DSD87, a $125 value.

Remote DSD87 (supports both Microsoft and Turbo C) . . . $395
Remote DSD87 OMF (includes Intel OMF support) $795

Link Link Link Link Link Link Link Link

ROM-Link is the first integrated link and locate utility to directly generate COM, EXE, SYS, HEX, or Intel OMF files. The only linker ever guaranteed faster than any present or future competitors. Supports Microsoft debugging information. Receive a FREE librarian and cross reference utility with your order!

ROM-Link $395

Soft Advances
10811 Washington Bl., Ste. 205 • Culver City, CA 90232 • (213) 559-7015

Number **33** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

mation of products is called a *convolution*.) The bias associated with the neuron in question must be added to the sum of the products. We now have a single (usually floating-point) value. We apply this value to the nonlinearity function, usually the sigmoid, and the result is the output value of that particular neuron.

This calculation is repeated for each neuron. Thus, $I*J$ multiplications and additions are required to perform the input-layer convolutions, along with J bias adds and J sigmoid calculations.

In step 3, the outputs of the input-layer neurons become, in effect, the input elements for the center-layer neurons. The computations are exactly the same as the input-layer computations. There are $J*K$ multiplications and additions plus K bias adds and K sigmoid calculations.

In step 4, the outputs of the center-layer neurons become the input elements for the output-layer neurons. The computations are exactly the same. There are $K*L$ multiplications and additions plus L bias adds and L sigmoid calculations. (For some problems—for example, *Is there a missile pad in this image?*—there will be only one output neuron; so $L = 1$.)

Here are steps 2 through 4 translated into BASIC:

```
for j = 1 to J
    sum = Ibias(j)
    for i = 1 to I
        sum = sum + Input(i)*Iwt(i,j)
    next i
    Iout(j) = 1/(1+exp(-sum))
next j

for k = 1 to K
    sum = Cbias(k)
    for j = 1 to J
        sum = sum + Iout(j)*Cwt(j,k)
    next j
Cout(k) = 1/(1+exp(-sum))
next k

for l = 1 to L
    sum = Obias(l)
    for k = 1 to K
        sum = sum + Cout(k)*Owt(k,l)
    next k
    Oout(l) = 1/(1+exp(-sum))
next l
```

The Learning Computations

To use a neural net, we need the precomputed values mentioned above. These precomputed values are obtained by training the neural net, which usually involves many training iterations. Each training iteration involves applying each of the exemplars in the training set to the neural net once and back-propagating the errors. The *learning computations* are the mathematical steps used to back-propagate the errors for one exemplar. The process uses the measured output errors from each exemplar to adjust the weights and biases of the various layers. (Remember, an exemplar is a data set whose desired neural net output is known.)

We have already defined I, J, K, and L. Now we define TS as the number of exemplars in the training set and ts as the corresponding array index.

We want the neural net to learn as quickly as possible. The back-propagation algorithm converges more quickly to a desired result if a momentum term is applied to the algorithm. Theoretically, the momentum term is optional; in practice, it is not. Regrettably, the use of the momentum term doubles the number of weight and bias arrays. (The weights and biases are the values that we compute during training.) For each weight and bias, the next value is calculated using both the current value and the previous value. The momentum term is the current value minus the previous value, times the momentum constant *alpha*.

After a training iteration, when the errors of all the exemplars have been back-propagated once, it might be nice to be able to examine the output errors that were obtained for all of the exemplars; otherwise, how can we tell when the net is trained? We will assign an array to hold this optional but useful data. And we will expand the input array to hold all of the exemplars, not just one input data set. If we don't expand the input array, we are going to have a serious disk I/O problem.

We have already used the convention of having the letters I, C, and O identify input, center, and output arrays. We will use N (for "now") to identify current values and P to identify previous values.

Please keep in mind that the following mathematical technique was known to be needed in the 1950s but was not invented until 1986. It is moderately complex. There are a fair number of data arrays. We will use the following names for the arrays:

```
Input(ts,i)   = the input elements
IwtN(i,j)     = the current input-layer weights
IwtP(i,j)     = the previous input-layer weights
IbiasN(j)     = the current input-layer biases
IbiasP(j)     = the previous input-layer biases
Iout(j)       = the input-layer outputs
pIdelta(j)    = part of the Idelta (error) calculation
CwtN(j,k)     = the current center-layer weights
CwtP(j,k)     = the previous input-layer weights
CbiasN(k)     = the current center-layer biases
CbiasP(k)     = the previous center-layer biases
Cout(k)       = the center-layer outputs
pCdelta(k)    = part of the Cdelta (error) calculation
OwtN(k,l)     = the current output-layer weights
OwtP(k,l)     = the previous output-layer weights
ObiasN(l)     = the current output-layer biases
ObiasP(l)     = the previous output-layer biases
Oout(l)       = the output-layer outputs
Target(ts,l)  = the known outputs of the exemplars
Error(ts,l)   = the optional output error array
```

As a necessary prelude to the learning computations, we must first perform recall by applying an exemplar to the neural net and performing the recall computations. The recall computations use only the current weight and bias values and ignore the previous ones. Once we have the various output values computed, we can begin the learning computations.

The learning computation is one part of the training process. The purpose of the training process is to obtain the precomputed weight and bias values. The following steps are involved in training the neural net. Step 1 is performed once; the other steps are performed once for each exemplar to complete a training iteration and repeated until the neural net is trained.

1. Assign the element and target values for each exemplar. Assign the convergence factor *nu* (typically 0.5–1.0) and momentum term *alpha* (typically 0.0–1.0). Assign random values to the present weights and biases. Zero all other values.

2. Perform the input-layer convolution, bias, and sigmoids.

3. Perform the center-layer convolution, bias, and sigmoids.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

4. Perform the output-layer convolution, bias, and sigmoids.

5. Determine the output errors and adjust the output weights and biases.

6. Determine the center errors, and adjust the center weights and biases.

7. Determine the input errors, and adjust the input weights and biases.

Steps 2 through 4 are essentially identical to the corresponding steps performed during recall, outlined previously. The learning computations involve steps 5 through 7. The method is called back propagation because we begin with the errors at the output and back-propagate the errors toward the input.

Let's look at steps 5 through 7 in more detail:

Step 5: Get the output-layer error and error term (*Odelta*) and update weights and bias while computing a portion of the center-layer error term:

```
initialize Cdelta()=0    (zero the entire array)
for l=1 to L
    Error(ts,l)=Target(ts,l)-Oout(l)
    Odelta=Oout(l)*(1-Oout(l))*Error(ts,l)
    for k=1 to K
        temp=OwtN(k,l)
        OwtN(k,l)=OwtN(k,l)+Nu*Odelta*Cout(k)
                  +alpha*(OwtN(k,l)-OwtP(k,l))
        OwtP(k,l)=temp
        pCdelta(k)=pCdelta(k)+Odelta*OwtN(k,l)
    next k
    temp=ObiasN(l)
    ObiasN(l)=ObiasN(l)+nu*Odelta+alpha*(ObiasN(l)-
              ObiasP(l))
    ObiasP(l)=temp
next l
```

Step 6: Get the center-layer error term and update weights and bias while computing a portion of the input-layer error term:

```
initialize Idelta()=0    (zero the entire array)
for k=1 to K
    Cdelta=Cout(k)*(1-Cout(k))*pCdelta(k)
    for j=1 to J
        temp=CwtN(j,k)
        CwtN(j,k)=CwtN(j,k)+Nu*Cdelta*Iout(j)
                  +alpha*(CwtN(j,k)-CwtP(j,k))
        CwtP(j,k)=temp
        pIdelta(j)=pIdelta(j)+Cdelta*CwtN(j,k)
    next j
    temp=CbiasN(k)
    CbiasN(k)=CbiasN(k)+Nu*Cdelta+alpha*(CbiasN(k)-
              CbiasP(k))
    CbiasP(k)=temp
next k
```

Step 7: Get the input-layer error term and update weights and bias:

```
for j=1 to J
    Idelta=Iout(j)*(1-Iout(j))*pIdelta(j)
    for i=1 to I
        temp=IwtN(i,j)
        IwtN(i,j)=IwtN(i,j)+Nu*Idelta*Input(ts,i)
                  +alpha*(IwtN(i,j)-IwtP(i,j))
        IwtP(i,j)=temp
    next i
    temp=IbiasN(j)
    IbiasN(j)=IbiasN(j)+Nu*Idelta+alpha*(IbiasN(j)-
        IbiasP(j))
    IbiasP(j)=temp
next j
```


Software Tools
For Programmers & Non-Programmers

★ NEW ★ VERSION **Opt-Tech Sort**™ ★ NEW ★ VERSION

Opt-Tech Sort is a high performance Sort/Merge/Select utility. It can read, sort and write a file faster than most programs can read the data. Example: 1,000 records of 80 bytes can be read sorted and a new file written in less than 9 seconds (IBM XT). Opt-Tech Sort can be used as a stand-alone program or Called as a subroutine to over 30 programming languages.

Numerous features are included. A partial list of features includes: The ability to process files of any size. Many filetypes are supported including Sequential, Random, delimited, Btrieve, dBASEII & III and many others. Up to 30 key fields can be specified over 16 different types of data are supported. Powerful record selection capability allows you to specify which records are to be included on your output. Record reformatting allows you to change the structure of your output record and to output special fields such as record numbers for use as indexes. **Newest Features: Faster than ever! LIM EMS support. User exits to pass or receive records. Less disk space required and much more!** MS-DOS $149.

On-Line Help™

On-Line Help allows you to easily add Help Windows to all your programs. On-Line Help is actually two help packages in one. You get BOTH Resident (pop-up) and Callable help systems.

The resident version allows you to add help to any system. Your Help system is activated when the "Hot Keys" that you specify are pressed. You can then chain between help windows in any manner you desire. Upon clearing the help windows you are right back into what ever program you left.

The callable version allows you to easily display help windows from your programs. A simple Call to the help system makes the window appear. On-Line Help is callable from over 20 different languages.

You have full control over the help window content, size, color and location. MS-DOS $149. Demo $10.

Scroll & Recall™

Scroll & Recall is a resident screen and keyboard enhancement. It allows you to conveniently scroll back through data that has gone off the top of your display screen. Up to 27 screens of data can be recalled or written to a disk file (great for documenting systems operations). Also allows you to easily recall and edit previously entered DOS commands without retyping.

Scroll & Recall is very easy to use. Its a resident utility that's always there when you need it. MS-DOS $69.

Visa, M/C, AMEX, Check, Money Order, COD or Purchase Orders accepted

To order or to receive additional information just call and receive immediate highly qualified attention!

Opt-Tech Data Processing
P.O. Box 678 / Zephyr Cove, NV 89448
(702) 588-3737

Number 16 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

If you're going to work with the back-propagation algorithm, then you're going to have to get very friendly with the above algorithm. If you do so, you'll quickly discover that the algorithm can be improved by algebraic transformations, parameterization, and dataflow analysis.

Why, then, was the back-propagation algorithm presented in the above form? Because the above form is the most convenient direct translation into BASIC of the more general equations below. The equations below are taken directly from box 6 on page 17 of Lippman's article (see references) and are modified only to correspond to the three-layer perceptron and to the notation that we have been using in this article. The equations used by Lippman can be traced in turn to Rumelhart, Hinton, and Williams' 1986 paper, which originally describes the back-propagation method. This paper is reprinted in *Neurocomputing: Foundations of Research,* (see references).

In step 5, *Odelta* appears to be a scalar value. In fact, *Odelta* is really *Odelta(l)*, since it is calculated inside the L loop. Similarly, *Cdelta* is really *Cdelta(k)*, and *Idelta* is really *Idelta(j)*. *Cdelta* and *Idelta* are computed in two stages.

Here is what we are really computing, in shorthand notation:

Step 5:

```
Dl = Ol * (1-Ol) * (TARGETl-Ol)
Wkl(t+1) = Wkl(t) + Nu*Dl*Ok +alpha*(Wkl(t)-
Wkl(t-1))
```

Step 6:

```
Dk = Ok * (1-Ok) * (sum of Dl*Wkl for all l)
Wjk(t+1) = Wjk(t) + Nu*Dk*Oj +alpha*(Wjk(t)-
Wjk(t-1))
```

Step 7:

```
Dj = Oj * (1-Oj) * (sum ofDk*Wjk for all k)
Wij(t+1) = Wij(t) + Nu*Dj*INPUTi +alpha*(Wij(t)-
Wij(t-1))
```

where *W* = a weight or bias, *O* = an output, and *D* = a delta.

Remember, a bias is treated as if it were a weight that is always multiplied by 1 rather than by an input (in the input layer) or an output (in the center and output layers). In step 6 above, the summation was really calculated in step 5 of the earlier algorithm, and in step 7 above, the summation was really calculated in step 6 of the earlier algorithm.

Nu is a convergence factor, and *alpha* is the momentum factor. *Alpha* should be a value between 0 and 1. We have had good luck using *nu* = 1 and *alpha* = 0.5. If these values are too small, the back-propagation training will converge to the desired state too slowly. If they are too large, the desired state will be approached quickly—and then overshot.

Note that we added a remembrance of the errors during one training iteration to step 5. This is not included in either of the references, but it is very useful.

Is Recall or Learning More Important?

To the newcomer to neural nets, the recall operation is obviously more important. After all, you have to train the net only once.

Wrong! The problem is that neural net art is not yet advanced to the point where we can say:

> For this problem you need this many input-layer neurons and that many center-layer neurons. Use 32-bit floating point for this problem. You want to use *nu* = 0.8 and *alpha* = 0.4. This particular problem is sensitive to the random values initially assigned to the weights; so be sure to run several training sessions with various initial conditions. If the net has not trained in 577 training iterations, you have a bogus set of initial conditions.

Instead, you're going to have to run a lot of training sessions trying different combinations of the above variables. You might want to try a variation or three on the network configuration, such as the Kohonen variant. You might want to try diddling the learning algorithm (good luck!).

The application will suggest the correct number of input elements and output-layer neurons—usually. You're going to run a lot of training sessions before you get your first real-world application successfully up and running. And your 4.77-MHz, 8088-based PC simply is not fast enough to do this work within your normal life expectancy. You need *fast* hardware!

Training Session Time

How complex a real-world problem can you tackle? How fast is your hardware? Although we cannot yet predict whether a particular real-world problem can be solved with a practical neural net, we *can* predict how long it will take to run a training session for a given number of inputs, neurons, and exemplars. The arithmetic is very simple.

When your hardware gets fast enough, it takes as long to do an FP load or store as it does to do an add or multiply. For this reason, FP loads and stores are counted as floating-point operations when the big boys benchmark their machines. One of the most common floating-point benchmark is the Linpack. The results of these benchmarks, usually expressed in *megaflops*, or *mflops* (millions of floating-point operations per second), are available for a wide range of computers. The operations performed while training neural nets are similar to the Linpack benchmark. So, if we know how many floating-point operations are required for a training session, we can predict how long it will take.

Like the Linpack benchmark, the neural net training spends most of its time in the inner loops of steps 2 through 7. Steps 2 and 7 have *I**J* iterations of the inner loop; steps 3 and 6 have *J**K* iterations, and steps 4 and 5 have *K**L* iterations. The number of FP operations in each loop depends on the details of the hardware (how many registers in the FP math chip) and of the chosen algorithm. However, this is a reasonable estimate of the number of FP operations to perform training one time for one exemplar:

```
FLOPs/exemplar = 14*I*J + 18*J*K + 18*K*L
```

During a training session, the above will be repeated for each exemplar. A typical training session will have hundreds of iterations. Let's use the number 300 as an approximation:

```
FLOPs/training session = 300*TS*(14*I*J +
        18*J*K + 18*K*L)
```

Okay, let's assume our problem requires a 64x64 retina; so *I* equals 4096. Assume *J* = 64, *K* = 64, and *L* = 64. Assume 64 exemplars. How long will it take to run a training session on your 0.84-mflops 386/Weitek machine? Well, you have to execute 7.33E10 FLOPs; at 0.84 mflops that will take 24 hours. If you had only one training session to

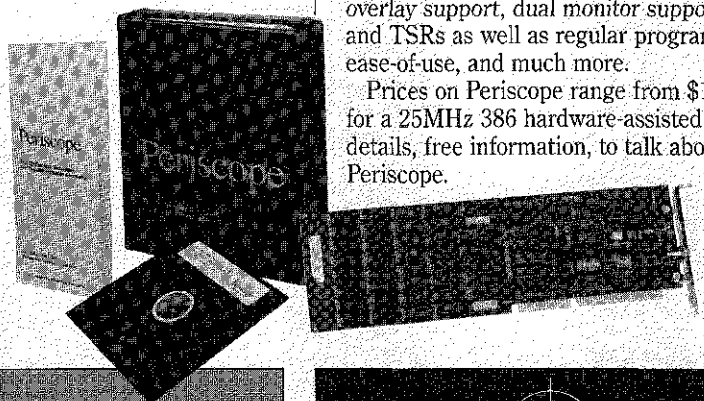
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

NEW...FROM PERISCOPE

Real-Time Debugging for 386s and 286s

User Jeff Garbers, Crosstalk Communications' Director of Software Development, has been debugging with Periscope® Model IV and says, "The hardware really makes Periscope shine, especially when you've got timing-related problems. I can now track down changing pointers and altered buffers on my 386. I've been using it to debug Crosstalk® Mk. 4 and there's just no better way to do it."

Periscope IV gives you the ability to debug time-sensitive programs, hardware-interrupt routines, and programs with intermittent errors. You can run your program at full speed while tracking down unwanted memory overwrites. You can use the information captured in Periscope IV's real-time trace buffer to see EXACTLY what the system is doing, and to improve its performance.



Periscope manual, disk, quick-reference card, and 512K Plus board that provides write-protected RAM for Periscope Models I, III, and IV.

A New Generation of Hardware-Assisted Debugging

Compatible with virtually any 286 or 386 with an AT-style bus, Periscope IV works on machines running up to 25MHz with any number of wait states. Because it gets information directly from the CPU, instead of from the system bus, Model IV is not sensitive to bus compatibility issues.

Periscope IV collects CPU information in its hardware trace buffer while the CPU runs at full speed. Whether you tell Periscope IV to capture just selected information or to capture everything, you can use its powerful trace buffer commands to search for and display the execution history the way you need to see it. And you can use the CPU cycle count information to get the last bit of performance out of your code.

With Periscope IV you can set hardware breakpoints on memory accesses (within the first 16MB), I/O ports, and data. You can also set breakpoints on the occurence of specific sequences of events, such as "watch for the routine FOO to begin executing, then while it is, watch for the variable BAR to be written." This capability, called sequential triggering, enables you to define complex conditions, then stop your program and examine what has happened when these conditions occur.

If you're developing a large application that needs all of the lower 640K, you can use the optional Plus board to keep Periscope totally out of normal DOS memory. The Plus board requires the use of a second slot.

The Periscope IV software is an extension of the software that comes with all models of Periscope. So, along with Periscope IV's powerful hardware, you get a full-function software debugger with source and symbol support for most popular PC compilers and linkers, Microsoft® Windows support, PLINK overlay support, dual monitor support, support for debugging device drivers and TSRs as well as regular programs, DOS independence, crash recovery, ease-of-use, and much more.

Prices on Periscope range from $145 for software-only Model II-X to $2,995 for a 25MHz 386 hardware-assisted Model IV. Call 800/722-7006 for pricing details, free information, to talk about your debugging needs, or to order your Periscope.

Order Your Periscope, Toll-Free, Today! 800-722-7006

MAJOR CREDIT CARDS AND QUALIFIED COMPANY PURCHASE ORDERS ACCEPTED

PC MAGAZINE The BEST of 1988

"No matter what your debugging needs, a single call to the Periscope Company has always sufficed..."

Best of 1988 "Development Tools" PC Magazine, January 17, 1989

The Periscope Company, Inc.

1197 PEACHTREE ST.
PLAZA LEVEL
ATLANTA, GA 30361
404/875-8080
FAX 404/872-1973

Number 6 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

run, that would be a very reasonable training time; if you're going to run dozens of training sessions, it's less reasonable. If you discover that you need 256 input-layer neurons, it takes four days per training session. What if you need hundreds of sessions to find a suitable—or optimum—network configuration?

Obviously, if all you have is an 80286/80287 AT clone, then you pick up your marbles and go home. What is really worrisome is that this problem is probably at the lower edge of real-world, image-related, neural net problems. What we really want is 150 or more mflops. At 150 mflops, that four-day training session drops to 32 minutes.

Don't Overload the Net

Don't ask a neural net to do everything. For instance, don't ask a neural net to recognize characters at all angles. Don't ask it to recognize characters of different sizes. In other words, preprocess the data.

If you're looking for particular kinds of objects in an image, and the objects have sharp edges, perform a two-dimensional FFT (see the sidebar on page 11) or the equivalent convolution to reduce the image to a wire frame (edge detection). A successful image-related neural net is going to need an image-processing front end and may be supervised by an expert system, with a human being (the real expert) supervising the entire operation.

Learning about Learning

We have learned a lot about neural net characteristics by applying hardware ranging from Atari STs to a 386/387 machine. (We are just beginning to work with a Sun 3/Sky Warrior array processor.) Obviously, the problems we have solved are not real world—we frankly call them "toy problems"—nevertheless, one can learn about neural nets and training with toy problems.

You can try optimizing the back-training algorithm. You learn what happens when you goof while modifying the algorithm. Amazingly, the net still learns if the goof is mild! You learn what happens when you mean to sort the exemplars but wind up sorting only the targets, which means you are asking the neural net to learn nonsense. The net learns the nonsense! You learn what happens when you use an inappropriately large value for *alpha*. The training will go dramatically haywire, like a lava lamp suddenly going into action.

You learn what happens when two or three layers of neurons are used. For some problems, two layers is better. You learn what happens when more and more neurons are added to a net. Performance improves up to a point, and then adding more neurons doesn't help.

If you want a net to learn to recognize a gizmo in four positions, assign a different output neuron for each position rather than asking a single output neuron to recognize all four positions. Both kinds of nets will learn, but the four-output net will learn a lot faster.

We discovered that presenting learning progress graphically was the only way to go. The fastest we have been able to run up to now is with a 386/387 machine running FORTRAN. However, the FORTRAN did not support graphics. We tried saving data arrays representing the training results onto disk, to be displayed graphically later using

NEW VERSION 2

MICROSOFT, TURBO AND MIX POWER C PROGRAMMERS...
C WINDOWS TOOLKIT™ PUTS YOU IN CHARGE OF VIDEO
and DATA-ENTRY

NEW FEATURES

- *Comprehensive data-entry and verification procedures with scanf( ) type syntax.*
- *Predefined data-entry functions for dates, SSNs, telephone numbers, currency amounts with commas, time, scientific notation, etc.*
- *Turn any window into an editor with your choice of keyboard mapping (Wordstar™ by default).*
- *Virtual windows up to available memory.*
- *Save windows to disk.*
- *Nested windows.*
- *Create and edit VGA/MCGA/EGA/Hercules fonts of any size.*

WINDOWING FUNCTIONS
- Create pop-up windows
- Create pull-down menus
- Create spreadsheet menus
- Create context-sensitive help screens
- Store windows in RAM or on disk
- Modify windows out of view
- Use 8 different types of exploding windows
- Scroll and move windows

MicroWay 386 SUPPORT*
- Runs in protected mode
- Uses 32-bit instruction set

*Extra cost option-call

SYSTEM CONTROL
- Device-independent output
- Supports VGA/MCGA/EGA/Hercules/CGA/MDA
- Detect the types of video adapter installed
- Switch between adapters
- Detect ANSI.SYS
- Control the size and position of the cursor
- Detect the enhanced keyboard
- Disable the video signal
- Delay program execution to microsecond resolution

VGA/EGA SUPPORT
- Use all 64 VGA/EGA colors
- Use VGA 28/43/50 line modes
- Use EGA 43-line mode
- Use 2 fonts simultaneously
- Design and edit custom fonts with FONTEDIT;* our font editor (included)
- Smooth scroll the screen
- Smooth pan the screen

FAST SCREEN I/O
- Write formatted output (like **printf( )**)
- Get snow-free output on the CGA
- Scroll the screen
- Use direct screen writes or the BIOS

HERCULES SUPPORT
- Create and edit your own Ramfonts™
- Detect the presence of a Hercules Card
- Detect Ramfont™ support
- Load and store Ramfonts™ to RAM or disk
- Switch between modes
- Detect whether the Hercules Card is in text or graphics mode

Full source code included — No run-time royalties.
Includes 230 page manual — 30-day Money-Back Guarantee

Magna Carta SOFTWARE

P.O. Box 475594
Garland, TX 75047-5594
FAX: (214) 226-0386

Requires: IBM PC, XT,AT, PS/2 or compatible.
Supports Microsoft C 5.0/5.1/Quick C, Borland Turbo C 1.5, Mix Power C

CALL (214) 226-6909

Only $99.95
(Texas Residents add 8% sales tax.)

Number **212** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Debug Multi-Megabyte DOS Programs

There was a time when the only way to debug very large DOS C programs meant suffering under 640KB memory constraints. Not any longer. *Instant-C* opens a new age of debugging under DOS.

Instant-C is specifically designed to debug large DOS programs. It works in "protected mode" on all 80286 and 80386-based PCs giving you access to **16MB** of extended memory. *Instant-C*'s source-level debugging lets you debug and run programs in excess of 100,000 lines of code without worrying about DOS memory limitations or resorting to overlays. Hard to believe? It's true. No other DOS debugger can live up to *Instant-C*'s standards.

And *Instant-C*'s debugging capabilities are unsurpassed. With features such as automatic run-time checking, data monitoring, source code animation, source code backtracing, and unlimited conditional breakpoints at your disposal, debugging your large programs will be faster than ever before.

Instant-C also includes an integrated C compiler, linker, editor, and source formatter. With *Instant-C* you get a powerful integrated tool package for working with your C programs. And *Instant-C* is compatible with your Microsoft C compiler.

Save yourself hours of debugging time every single day. Try *Instant-C* and discover how to make debugging your large DOS C programs quicker than you ever believed possible. With *Instant-C*'s **30 day no risk money back guarantee** the only thing you've got to lose is bugs.

- **Debug Multi-Megabyte Programs**
- **Automatic Run-time Checking**
- **All Debugging in C**
- **Unlimited Conditional Breakpoints**
- **Compatible with your Assembly Code**
- **Source Code Backtracing**
- **Support for Full C Language**
- **Data Breakpoints**
- **Source Code Animation**
- **Compatible with Microsoft C**

Rational Systems' 30 Day No Risk Money Back Guarantee

If for any reason you decide not to keep *Instant-C*, just return it to us within 30 days of purchase and we will gladly refund your money.

Order *Instant-C* today!

***only* $795**

Rational
Systems, Inc.

220 North Main Street
Natick, MA 01760
Tel: **(508) 653-6006**
FAX: **(508) 655-2753**

Trademarks are acknowledged to Rational Systems, Inc., and Microsoft Corp.

Number 20 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Marvin Minsky showed up at the First International Conference on Neural Nets (ICNN) in 1987 and was roundly booed. His reception at the 1988 ICNN was much friendlier, with some good-natured ribbing. Translation: Neural nets were securely reestablished, and winners are often gracious.

Turbo C. This is a pain, and it does not allow for continuous graphical updates.

Miscellany

The sigmoid is favored because lots of nonlinear functions have been tried and the sigmoid works best—so far. It is a continuously differentiable function, and one whose value is continuously variable; yet most of the slope is near an argument value of 0. The value of the sigmoid cannot be less than 0 or greater than 1.

As mentioned earlier, Figure 2 is a plain, generic neural net. Various researchers use modifications of this basic topology. Neurons—sorry, make that summer/nonlinear elements—may feed back to earlier layers or may feed sideways to elements in the same layer. Special binary forms of nets may be used (the Hopfield or Hamming nets). Lippman does a very good job of covering the six most common types of nets.

Conclusion

We have confined our discussion to digital implementations of neural nets, because those are the kind you can actually make today as practical hardware. Others—Carver Mead, for example—are working on analog implementations. The work JPL is doing involves one-time switchable thin-film resistors on amorphous silicon. Note the phrase *one-time*. On the other hand, a square yard of amorphous silicon is relatively inexpensive. You can put an awful lot of resistors on a square yard of silicon using submicron design rules.

We believe digital neural nets are going to win, for the same reasons that modems are now made using digital filters while ten years ago they were all made using analog filters. There is nothing like the power of dedicated VLSI, and nothing like the economy of massive production once some real-world applications take hold. Integration might even solve the learning problem (massive calculations needed). For instance, Bellcore already has a dedicated neural net chip that has built-in provisions for learning. (When the Baby Bells were spun out of AT&T, they lost access to Bell Labs; they formed a consortium and funded Bellcore.) Bellcore asserts that the chip "learns" 100,000 times faster than a computer simulation (presumably 4.77-MHz 8088, not a Cray Y-MP).

While we're waiting for production quantities of something like the Bellcore chip, we can use dataflow machines, which are roughly two orders of magnitude more cost-effective than general-purpose computers. A dataflow machine contains some fast memory sequencers and a math chip for each symbol in the equation being solved. (The FFT boys have used dataflow machines for years—ask Weitek.)

We think a $10,000 board-level product could be built that would perform training at 10 million connections per second, equivalent to 150 single-precision Linpack mflops. A $100,000 tabletop dataflow machine would train at 100 million connections per second. (Ask a neural net researcher how he or she would like a machine that would train at 100 million connections per second!) One company—NEC in Japan—has announced plans to build such a machine. PJ

REFERENCES

Lippmann, R. P., "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, April 1987.

Anderson, J. A., and Edward Rosenfeld, *Neurocomputing: Foundations of Research*, The MIT Press, Cambridge, Mass., 1988.

Tom Waite is leader of algorithm development at Viacom Systems, Inc., in San Jose, California. He has a background in applied math and neurophysiology and has worked in the area of digital signal and image processing for the past seven years.

Hal W. Hardenbergh is a senior hardware engineer who sometimes programs. Hal also works at Viacom Systems, Inc.

The code disk for this issue includes a complete implementation of the back-propagation algorithm in BASIC.

Logic Gem $198

AT LAST!

Develop perfect logic

- Automatically completes incomplete logic
- Removes redundancies
- Eliminates contradictions
- Sorts and optimizes logic

Generate bug-free source

- C
- Pascal
- dBASE
- Fortran
- English
- Structured Basic
- Interpretive Basic

Sterling Castle
(800)722-7853
(800)323-6406 CA

VISA • MC • AE
90 Day Money Back Guarantee

Download demo • (213) 453-7705 • 3/12/24,8,N,1

Number 268 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WHETHER REPORT.

Whether you're a software developer writing new applications for the IBM or Mac, or a PC user securing proprietary data files, software and data protection has never had a brighter silver lining. For a number of very good reasons.

Beginning with the 'whether-expert' Rainbow Technologies. And ending with its Software Sentinel family of hardware keys. Starring five models that fit virtually any software program or data file you need to protect.

There's the best-selling SentinelPro for the IBM PC/XT/AT, PS/2 and compatibles, and even the Atari ST. Known worldwide for its virtually unbreakable security. And its ASIC technology. And its invisible operation. A close relation, the Sentinel-C stands at-the-ready for custom configurations and multiple software packages.

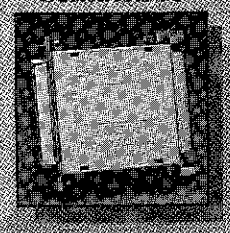
In the Apple market, security-minded Mac software developers turn to Eve. For completely transparent operation and world-class security of the protected software. Just by plugging Eve into the Mac ADB connector.

PC users wanting a low cost, user-friendly solution to the problem of securing sensitive data can call on the DataSentry. Using a proprietary Rainbow algorithm or DES, the DataSentry encrypts data files on individual PCs, protects modem transmissions and secures data on local area networks.

Rainbow's latest protection strategy is the SentinelShell–that lets users place a 'shell' around existing, off-the-shelf programs. Because access can be limited to those issued a key, libraries, universities and corporations can very simply guard their software investments.

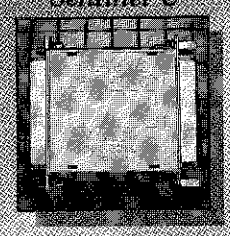
Whatever your whether, Rainbow Technologies has the software and data protection products that make the difference. For more information, call 714-261-0228 in the U.S., or contact Rainbow Technologies Ltd. in the United Kingdom for the distributor nearest you. Whethercasters are standing by.

SentinelPro™



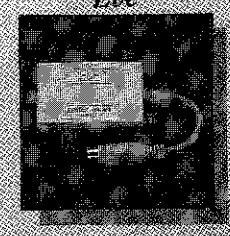
• Runs under DOS, OS/2 and Xenix • Algorithm technique (Never a fixed response) • External parallel port installation • Minimal implementation effort • Higher level language interfaces included • 100 times faster than fixed-response devices (1 ms) • ASIC design for reliability

Sentinel-C™



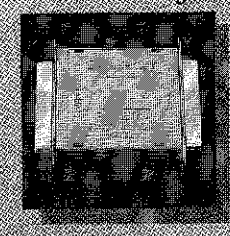
• Protects multiple packages with one device • 256 bytes of non-volatile memory programmed before shipment of the software • Rainbow supplies a unique adapter for programming the unit • Higher level language interfaces included • Runs under DOS, OS/2 and Xenix • External parallel port installation

Eve™



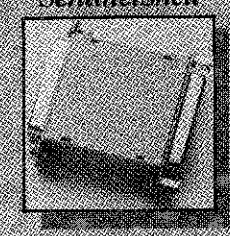
• For the Macintosh SE and II • Complies with Apple Desktop Bus Interface requirements • Rainbow-assigned developer passwords to prevent tampering by other developers or sophisticated 'hackers' • 7 locks per key, usable individually or in combination, on one or up to seven applications

DataSentry™



• Completely user-installable • Packet-sized external device • Menu-driven, user-friendly interface • Single- or multi-user security system • Audit trail, log-on identifiers and automatic encryption/decryption of entire directories • Secures data transmitted by modems • Prevents recovery of data by utility programs

SentinelShell™



• Runs under DOS on IBM PCs and compatibles • Protects without requiring access to the source code • Completely transparent to the end user • User-friendly software • Packet-size key attaches quickly to any standard PC parallel port • ASIC design for reliability

RAINBOW TECHNOLOGIES

18011-A Mitchell South, Irvine, CA 92714 • (714) 261-0228 • TELEX: 386078 • FAX: (714) 261-0260
Rainbow Technologies, Ltd., Shirley Lodge, 470 London Rd., Slough, Berkshire, SL3 8QY, U.K., Tel: 0753-41512, Fax: 0753-43610

© 1989 Rainbow Technologies. All product names are trademarks of their respective manufacturers.

PRODUCED 2004 BY UMZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Fast Line Drawing for the EGA and VGA

by
Michael Abrash

Installment 12: In which we draw lines like a bat out of . . .

This article was originally written for Turbo Technix *magazine. However, this is the first time this article has appeared in print.*

This installment of "On Graphics" is a bit different from the norm. Some time ago I wrote an article about fast line drawing for the late, lamented *Turbo Technix*, the Borland magazine. The article explained Bresenham's line-drawing algorithm in some detail and presented a clear and reasonably fast EGA/VGA implementation of the algorithm in C so that anyone with a mind to could understand how the process of drawing a line worked. Then, with the theoretical and educational aspects taken care of, the article presented a *very* fast assembler implementation of the algorithm. (If something is worth doing, it's worth doing right, and lines should be drawn with assembler code!)

Pleased with the result, I shipped the article off to *Turbo Technix*, where it was to be published in the seventh issue. Alas, issue six was the last of its line, and there I was, with an article I was sure a good many people would be interested in reading . . . and no way to get it into print.

Well, thanks to cooperation from the people at Borland and *PJ*, I'm able to run the article here. It's not in the usual "On Graphics" style, since it was written for a different audience. It spends more time explaining details, such as the organization of display memory, that most of you know like the back of your hand by now. Still, a little extra detail never hurts, and even if all you look at is the assembler code in Listing 3, it *is* good code. But if you want to know whether it's the fastest Bresenham's implementation possible, I must tell you that it isn't.

First, the code could be sped up a bit by shuffling and combining the various error-term manipulations, but that results in truly cryptic code. I wanted you to be able to relate the original algorithm to the final code, so I skipped those optimizations. Also, write mode 3, which is unique to the VGA, could be used for faster drawing. (For information on write mode 3 and other EGA/VGA details discussed in this article, see previous installments of "On Graphics" in *PJ*.)

Second, horizontal, vertical, and diagonal lines could be made special cases, since those particular lines require little calculation and so can be drawn very rapidly. This is especially true of horizontal lines, which can be drawn eight pixels at a time.

Third, in-line code or duplicated code (or both) could be used to eliminate most of the branches in Listing 3. Because the 8088 is notoriously slow at branching, that would make quite a difference in overall performance. (If you're interested in in-line code and similar assembler techniques, watch for my book on high-performance programming, *The Zen of Assembler*, due this summer from Scott, Foresman.)

Even if I didn't know that there were further optimizations to be made to my line-drawing implementation, I'd *assume* that there were. As I'm sure the experienced programmers among you know, there are dozens of ways to tackle any problem in assembler, and someone else always seems to have come up with a trick that never occurred to you. I've incorporated a suggestion made by Jim Mackraz in the code in this article, and I'd be interested in hearing of any other tricks or tips you readers may have. The line-drawing implementation in Listing 3 is fast enough for most purposes, though, so let's get to it.

Line Drawing

There are two important characteristics of any line-drawing function. First, it must draw a reasonable approximation of

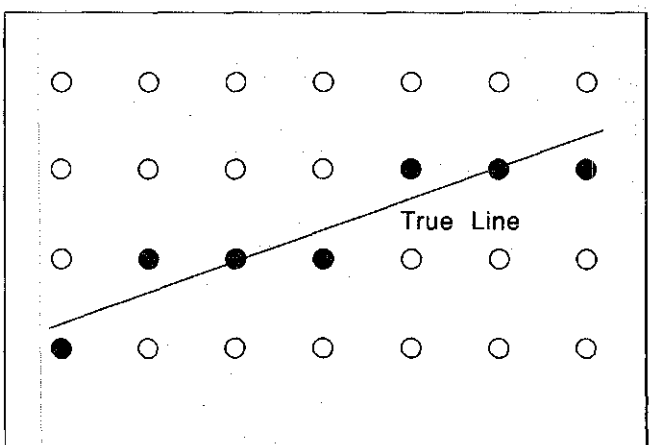
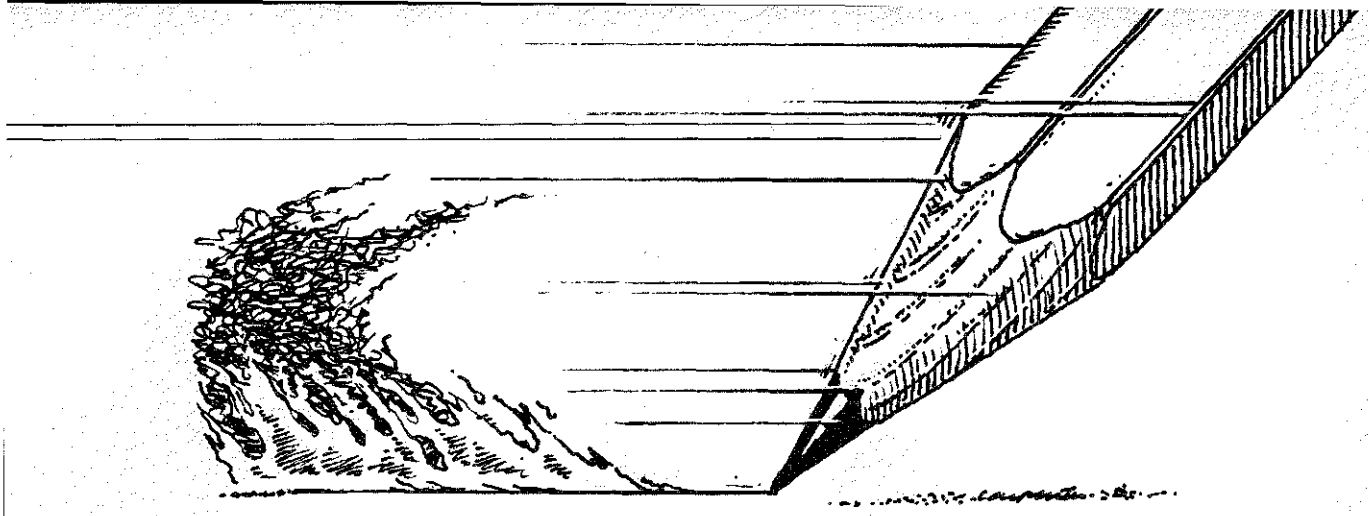


Figure 1. Approximation of a true line on a finite-resolution graphics display. Black circles are screen dots turned on to approximate the true line; white dots are screen dots that are not turned on.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



a line. Since a computer screen has limited resolution, a line-drawing function must approximate a straight line by drawing a series of dots in what actually amounts to a jagged pattern that generally proceeds in the desired direction. That pattern of dots must reliably suggest to the human eye the true line it represents. Second, to be usable, a line-drawing function must be *fast*. Minicomputers and mainframes generally have hardware that performs line drawing, but most microcomputers offer no such assistance. When drawing lines on the CGA, EGA, or VGA, the PC's CPU must draw lines on its own, and, as many users of graphics-oriented software know, that can be a slow process indeed.

Line-drawing quality and speed derive from two factors: the algorithm used to draw the line and the implementation of that algorithm. In this article, I will present two implementations for the EGA and VGA (henceforth referred to collectively as the EVGA) of Bresenham's line-drawing algorithm, an algorithm that provides adequate line quality and excellent drawing speed. (Bresenham also has circle-drawing and ellipse-drawing algorithms.) The first implementation, written in Turbo C, illustrates the workings of the algorithm and draws lines at a good rate. The second implementation, written in assembly language and callable directly from Turbo C, draws lines at an extremely high speed, four to six times faster than the C version. Between them, the two implementations illuminate Bresenham's line-drawing algorithm and provide high-performance line-drawing capability.

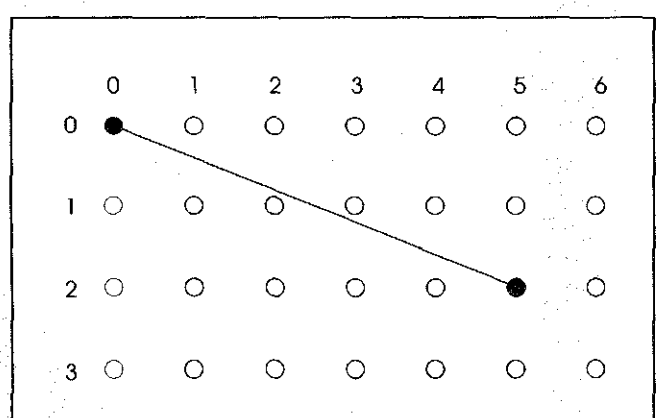


Figure 2. Drawing a line from (0,0) to (5,2). Only the endpoints fall squarely on a dot on the screen. All other points must be approximated by drawing nearby dots.

What's So Tough about Drawing a Line?

The difficulty in drawing a line lies in generating a set of dots that, taken together, are a reasonable facsimile of a true line. Only horizontal, vertical, and diagonal lines can be drawn precisely along the true line being represented; all other lines must be approximated from the array of dots that a given video mode supports, as shown in Figure 1.

Considerable thought has gone into the design of line-drawing algorithms, and a number of techniques for drawing high-quality lines have been developed. (See references.) Unfortunately, most of these techniques were developed for powerful, expensive graphics workstations and require very high resolution, a large color palette, or floating-point hardware. These techniques tend to perform poorly and produce less visually impressive results on PCs.

Bresenham's line-drawing algorithm, on the other hand, is uniquely suited to microcomputer implementation in that it requires no floating-point operations, no divides, and no multiplies inside the line-drawing loop. Moreover, it can be implemented with surprisingly little code.

Bresenham's Line-Drawing Algorithm

The key to grasping Bresenham's algorithm is to understand that when you draw an approximation of a line on a finite-resolution display, each dot drawn will lie either exactly on the true line or to one side or the other of the true line. The amount by which the dot deviates from the true line is the error of the line drawing at that point. As the drawing of the line progresses from one dot to the next, the cumulative error can be used to tell when, given the resolution of the display, a more accurate approximation of the line can be drawn by placing a given dot one unit of screen resolution away from its predecessor in the horizontal direction, the vertical direction, or both.

Let's examine the case of drawing a line where the horizontal, or x, length of the line is greater than the vertical, or y, length and both lengths are greater than 0. For example, suppose we are drawing a line from (0,0) to (5,2), as shown in Figure 2. Note that Figure 2 shows the upper left corner of the screen as (0,0), rather than (0,0) being at the more traditional lower left corner location. Because of the way in which the PC's graphics are mapped to memory, it is simpler to work within this framework, although a translation of y from increasing downward to increasing upward could be effected easily enough by simply subtracting the x coordinate from the screen height minus 1. (If you are more comfortable with the traditional coordinate system, feel free to modify the code in listings 1 and 3.)

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

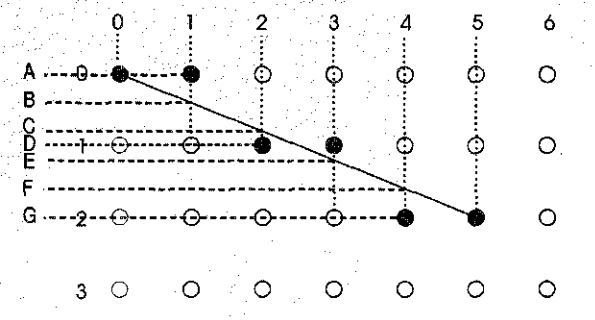


Figure 3. When drawing a line from (0,0) to (5,2), Bresenham's algorithm uses the error term to decide at which *y* coordinate the closest approximation to the line for each *x* coordinate lies.

In Figure 2, the endpoints of the line fall exactly on displayed dots. However, no other part of the line squarely intersects the center of a dot, meaning that all other dots will have to be plotted as approximations of the line. The approximation that Bresenham's algorithm takes is to move exactly one dot along the major dimension of the line each time a new dot is drawn while moving one dot along the minor dimension each time the line moves more than halfway between dots along the minor dimension.

In Figure 2, the *x* dimension is the major dimension. This means that six dots—one each at *x* coordinates 0, 1, 2, 3, 4, and 5—will be drawn. The trick, then, is to decide on the correct *y* coordinates to accompany these *x* coordinates.

It's easy enough to select the *y* coordinates by eye in Figure 2 on the basis of the *y* coordinate closest to the line for each *x* coordinate. The appropriate *y* coordinates are 0, 0, 1, 1, 2, and 2. Bresenham's algorithm makes the same selections, based on the same criterion, by keeping a running record of the error of the line—that is, how far from the true line the current *y* coordinate is—at each *x* coordinate, as shown in Figure 3. When the running error of the line indicates that the current *y* coordinate deviates from the true line to the extent that the adjacent *y* coordinate would be closer to the line, then the current *y* coordinate is changed to that adjacent *y* coordinate.

Let's take a moment to follow the steps Bresenham's algorithm would go through in drawing the line in Figure 3. The initial dot is drawn at (0,0), the start point of the line. At this point the error of the line is 0.

Since *x* is the major dimension, the next dot has an *x* coordinate of 1. The *y* coordinate of this dot will be either 0 (the last *y* coordinate) or 1 (the adjacent *y* coordinate in the direction of the end point of the line), depending on which is closer to the true line at this *x* coordinate. The running error at this point is B minus A. Since this amount is less than 1/2 (that is, less than halfway to the next *y* coordinate), the *y* coordinate does not change at $x = 1$. Consequently, the second dot is drawn at (1,0).

The third dot has an *x* coordinate of 2. The running error at this point is C minus A, which is greater than 1/2 and therefore closer to the next *y* coordinate than to the current one. The third dot is drawn at (2,1), and 1 is subtracted from the running error to compensate for the adjustment of one dot in the current *y* coordinate. The running error of the dot actually drawn at this point is C minus D.

The fourth dot has an *x* coordinate of 3. The running error at this point is E minus D; since this is less than 1/2, the current *y* coordinate doesn't change. The fourth dot is drawn at (3,1).

The fifth dot has an *x* coordinate of 4. The running error at this point is F minus D; since this is greater than 1/2, the current *y* coordinate advances. The third dot is drawn at (4,2), and 1 is subtracted from the running error. The error of the dot drawn at this point is G minus F.

Finally, the sixth dot is the end point of the line. This dot has an *x* coordinate of 5. The running error at this point is G minus G, or 0, indicating that this point is squarely on the true line—as of course it should be, given that it's the end point—so the current *y* coordinate remains the same. The end point of the line is drawn at (5,2), and the line is complete.

Conceptually, that's really all there is to Bresenham's algorithm. The algorithm is a process of drawing a dot at each possible coordinate along the major dimension of the line, in each case at the closest possible coordinate along the minor dimension. The running error is used to keep track of when the coordinate along the minor dimension must change to remain as close as possible to the true line. Bresenham's algorithm is particularly fast because it allows the running error to be maintained without any divides, multiplies, or floating-point arithmetic, as we'll see in Listing 1.

While we've discussed line drawing in only one of the octants, the above description of the case where *x* is the major dimension, *y* is the minor dimension, and both dimensions are greater than 0 can be readily generalized to all eight octants in which lines could be drawn—and we'll do just that in Listing 1.

Now, I realize full well that the above is nothing more than a seat-of-the-pants overview of Bresenham's line-drawing algorithm. Given the "working-code" orientation of *PJ* readers, I'm going to leave the explanation at that, especially since a full explanation would require many pages and a cryptic mathematical derivation. If you want or need a full mathematical treatment, refer to pages 433–436 of Foley and Van Dam's *Fundamentals of Interactive Computer Graphics* (see references). This source provides the derivation of the integer-only, divide-free version of the algorithm as well as Pascal code for drawing lines in one of the eight possible octants.

Strengths and Weaknesses

The overwhelming strength of Bresenham's line-drawing algorithm is speed. With no divides, no floating-point operations, and no need for variables that won't fit in 16 bits, it is perfectly suited for IBM PCs and compatibles.

The weakness of Bresenham's algorithm is that it produces relatively low quality lines by comparison with most other line-drawing algorithms. In particular, lines generated with Bresenham's algorithm can tend to look somewhat jagged. On the PC, however, jagged lines are an inevitable consequence of relatively low resolution and a small color set; so lines drawn with Bresenham's algorithm don't look all that much different from lines drawn in other ways. Besides, in most applications users are far more interested in the overall picture than in the primitive elements the picture is built from. As a general rule, any collection of dots that trend from point A to point B in a straight fashion is accepted by the eye as a line.

Bresenham's algorithm is successfully used by many current PC programs, and by the standard of this wide

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

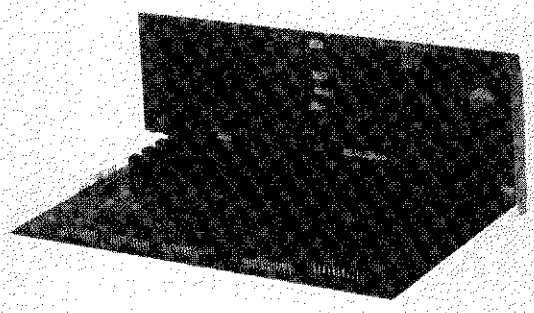
VERY HIGH PERFORMANCE

Processors, Memory, and Display Adapters

The X24 High performance processor

- 12 or 16 MHz 80286 with NO WAIT STATES!
- Small size ("XT" height and length) passive bus design
- 1 to 4 Mbyte 0 wait state dynamic memory
- Fully "AT" compatible Award BIOS
- Runs DOS versions 2.2 and later, Xenix and OS/2

The X24 combines the best of motherboard and backplane designs in a 100% AT compatible system. Incorporating a 16 MHz 80286, the X24 processor is designed to operate with the PC Tech Advanced System Motherboard, which contains the peripheral interfaces (hard disk, floppy disk, two serial ports and a parallel port). The X24 processor can also be used with other totally passive bus backplanes. Most critical components including the microprocessor and up to 4 megabytes of fast memory are contained on a single PC size plug-in card. This allows the processor and main system memory to be serviced or upgraded without disturbing other peripherals such as serial ports and disk drives.



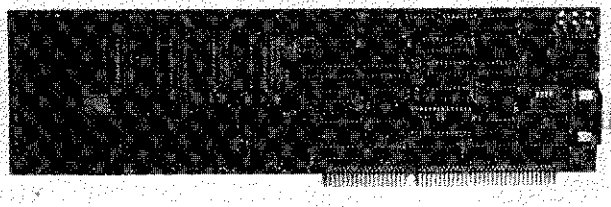
PC Tech X24 and ASMB

The PC Tech Advanced System Motherboard

- Built in "IDE" interface for AT interface type hard drives
- Fully AT compatible floppy disk support for 3.5", 5.25" drives, capacities of 360k, 1.2m and 1.44m
- Two serial ports and one parallel port
- 8 total expansion slots PC/XT/AT compatible (4 slots have 32 bit bus)

The PC Tech Advanced System Motherboard is designed to complement PC Tech's X24 and X32 high performance processor cards. It contains the mass storage interfaces necessary for a complete system, plus the basic I/O required in most systems. Extra care has been given to FCC compliance by design.

34010 Monochrome Graphics Adapter II



PC Tech Mono-II

- Up to 384k bytes display memory
- Up to 2 Megabytes program memory
- Software is RAM based, allowing complete operating software replacement and timing re-programming from the host bus
- 34010 program loader included. Assembler, debugger, and C compiler available.
- Full hardware and software CGA, MDA and Hercules emulation
- Single bit shared memory bit-map with optional resolution up to 2048 x 1536 (736 x 1008 standard)
- Very high resolution COLOR version available
- Custom 34010 software development available

The TMS34010 is a true general purpose graphics processor. PC Tech makes the total processing power of the 34010 available to both programmers and end users. Our 34010 Monochrome Graphics Adapter is designed to allow programming from the PC/XT/AT host bus. You can completely replace our 34010 software with yours to directly harness the incredible image processing power of the TMS 34010 for your application. We make a complete set of development tools available, including an assembler, C compiler, program loader, 34010 debugger, and PC interface tracer/debugger. Our standard product includes support for extended CGA, MDA and Hercules emulation as well as a host addressable graphics bit- map. We also support and recommend the DGIS graphics interface standard (from Graphic Software Systems) for applications development as an alternative to native 34010 software development. Ready to run drivers are available for most major applications software packages as well.

Custom Designs Available

PC Tech will license most products for non-exclusive manufacture. We will also customize any of our designs to better meet your needs on our in-house CAD systems. All of our standard products are available in private label versions.

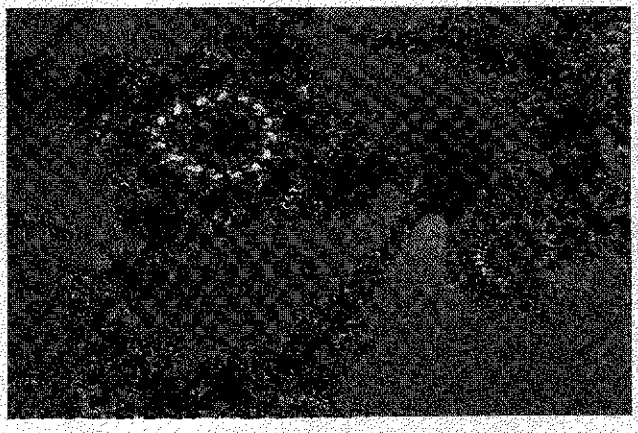
About PC Tech

PC Tech has been designing, manufacturing and marketing high performance PC related products for over three years. Our standard product line includes processor, memory, and video products. All products are designed, manufactured and supported in our Lake City, Minnesota facilities.

Designed, Sold and Serviced By:

PC Tech

907 N. 6th St., Lake City, MN 55041
(612) 345-4555 • (612) 345-5514 (FAX)



High resolution fractal produced on the PC Tech COLOR 34010

Number **208** on the Reader Service Card

PC, XT, AT, DGIS, Hercules, and Award BIOS are trademarks or registered trademarks of their respective companies.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED.

acceptance, the algorithm is certainly good enough. Then, too, users hate waiting for their computers to finish drawing. By any standard of drawing performance, Bresenham's algorithm excels.

A Turbo C Implementation of Bresenham's Algorithm

Listing 1 is a Turbo C implementation of Bresenham's line-drawing algorithm for modes 0EH, 0FH, 10H, and 12H of the EVGA, called as function **EVGALine**. Listing 2 is a sample program to demonstrate the use of **EVGALine**.

The **EVGALine** function itself performs four operations. **EVGALine** first sets up the EVGA's hardware so that all dots will be drawn in the desired colors. This is accomplished by setting two of the EVGA's registers: the Enable Set/Reset register and the Set/Reset register. Setting the Enable Set/Reset to the value 0FH, as is done in **EVGALine**, causes all drawing to produce dots in the color contained in the Set/Reset register. Setting the Set/Reset register to the passed color, in conjunction with the Enable Set/Reset setting of 0FH, causes all drawing done by **EVGALine** and the functions it calls to generate the passed color. In summary, setting up the Enable Set/Reset and Set/Reset registers in this way causes the remainder of **EVGALine** to draw a line in the specified color.

EVGALine next performs a simple check in order to cut in half the number of line orientations that must be handled separately. Figure 4 shows the eight possible line orientations among which a Bresenham's algorithm implementation must distinguish. (Assume that lines radiate outward from the center of the figure, falling into one of eight octants delineated by the horizontal and vertical axes and the two diagonals.) The need to identify these line orientations falls out of the nature of the algorithm; the orientations are distinguished by which coordinate forms the major axis and by whether *x* and *y* each increases or decreases from the line start to the line end.

A moment of thought will show, however, that four of the line orientations are redundant. Each of the four orientations for which *DeltaY*, the *y* component of the line, is less than 0 (that is, for which the line-start *y* coordinate is greater than the line-end *y* coordinate) can be transformed into one of the four orientations for which the line-start *y* coordinate is less than the line-end *y* coordinate simply by reversing the line start and end coordinates so that the line is drawn in the other direction. **EVGALine** does this by swapping (*x*0,*y*0) (the line-start coordinates) with (*x*1,*y*1) (the line-end coordinates) whenever *y*0 is greater than *y*1.

With this accomplished, **EVGALine** still must distinguish among the four remaining line orientations. Those four orientations form two major categories: orientations for which the *x* dimension is the major axis of the line and orientations for which the *y* dimension is the major axis. As shown in Figure 4, octant 1 (where *x* increases from start to finish) and octant 2 (where *x* decreases from start to finish) fall into the latter category and differ in only one respect: the direction in which the *x* coordinate moves when it changes. Handling of the running error of the line is exactly the same value for both cases—as one would expect given the symmetry of lines differing only in the sign of *DeltaX*, the *x* coordinate of the line. Consequently, for those cases where *DeltaX* is less than 0, the direction of *x* movement is made negative, and the absolute value of *DeltaX* is used for error-term calculations.

Similarly, octants 0 (where *x* increases from start to finish) and 3 (where *x* decreases from start to finish) differ only in the direction in which the *x* coordinate moves when it changes. The difference between line drawing in octants 0 and 3 and line drawing in octants 1 and 2 is that in octants 0 and 3, since *x* is the major axis, the *x* coordinate changes on every dot of the line and the *y* coordinate changes only when the running error of the line dictates. In octants 1 and 2, the *y* coordinate changes on every dot and the *x* coordinate changes only when the running error dictates, since *y* is the major axis.

There is one line-drawing function for octants 0 and 3, **Octant0**, and one line-drawing function for octants 1 and 2, **Octant1**. A single function with **if** statements could certainly be used to handle all four octants, but at a significant performance cost. There is, on the other hand, very little performance cost to grouping octants 0 and 3 together and octants 1 and 2 together, since the two octants in each pair differ only in the direction of change of the *x* coordinate.

EVGALine determines which line-drawing function to call and with what value for the direction of change of the *x* coordinate on the basis of two criteria: whether *DeltaX* is negative or not and whether the absolute value of *DeltaX* (|*DeltaX*|) is less than *DeltaY* or not, as shown in Figure 5. Recall that the value of *DeltaY*, and hence the direction of change of the *y* coordinate, is guaranteed to be non-negative as a result of the earlier elimination of four of the line orientations.

After calling the appropriate function to draw the line (more on those functions shortly), **EVGALine** restores the state of the Enable Set/Reset register to its default of 0. In this state, the Set/Reset register has no effect, so it is not necessary to restore the state of the Set/Reset register as well. **EVGALine** also restores the state of the Bit Mask register (which, as we will see, is modified by **EVGADot**, the dot-drawing routine actually used to draw each dot of the lines produced by **EVGALine**) to its default of 0FFH.

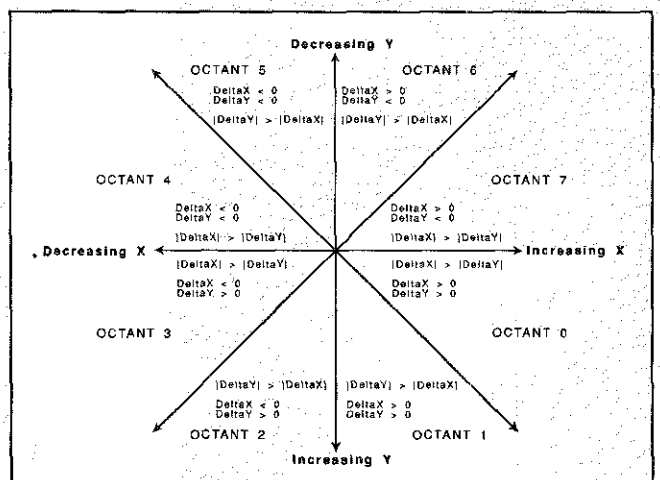


Figure 4. The eight line orientations among which the implementation of Bresenhams' algorithm must distinguish. Lines in the four upper octants are converted to equivalent lines in the four lower octants by a switching of the start and end coordinates. Lines start at the center of the figure and radiate outward.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

GRAPHICS

Break Out of the Text Barrier with your Herc, CGA, EGA, or VGA

PCX PROGRAMMER'S TOOLKIT **$125**

Over 55 routines to display, save, print PCX bitmapped graphics - Fast, easy to use - Display windows or full screen - Image library manager - use images from Paintbrush, Scanners, Clip-Art, or Capture Screens - No Royalties !

PCX TEXT **$99**

Display text in any graphics mode - Includes 12 fonts - Create custom fonts with dual font editor - User input routines - Font scaling - Additional fonts available.

PCX F/X **$99**

Add Special Effects ('FX')! Fade, wipe, push, roll, slide, split, crush, spiral, explode your graphics! Palette manipulation - Animation - Sound Effects

All packages support 12 compilers for BAS, PAS, C, FOR, ASM and Clipper.

Source available.

G·E·N·U·S

MICROPROGRAMMING

11315 Meadow Lake • Houston, Texas 77077 • (713) 870-0737

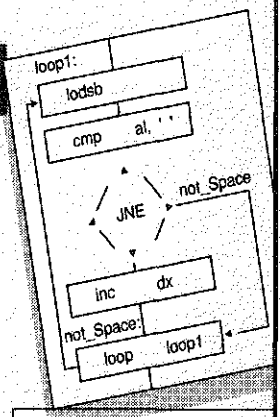
800•227•0918

Number **261** on the Reader Service Card

ASMFLOW $99.95

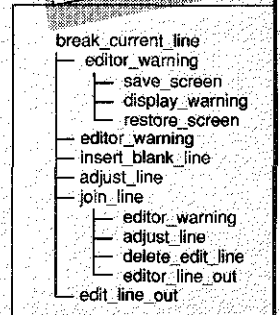
AT LAST!

YOU CAN STEP BACK AND TAKE A LOOK AT YOUR CODE



FLOW CHART AND ANALYZE YOUR ASSEMBLY LANGUAGE SOURCE CODE

- Flow Charts
- Tree Diagrams
- Stack Sizing
- Register Analysis
- CPU Timing Analysis
- Procedural X-Reference
- 8088/87 to 80386/387
- Context-Sensitive Help
- Menu/Batch/Command line Operation
- MASM 5.1 Compatible



QUANTUM SOFTWARE

19855 Stevens Creek Blvd, Suite 154
Cupertino, CA 95014
(408) 244-6826 – Free Demo

VISA • MC 30 - Day Money Back Guarantee

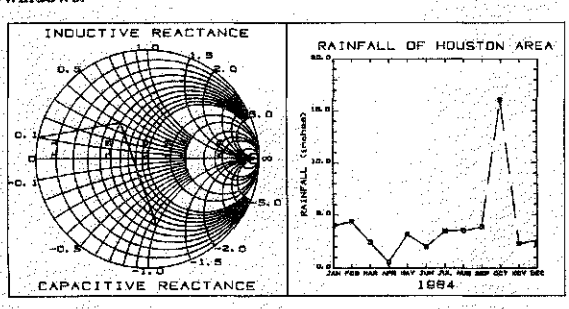
Number **252** on the Reader Service Card

INGRAF 2.00

A multi-device graphics library for Scientific, Engineering, and Business users! Supports video, printer, and plotter graphics for personal computers.

Over 100 routines to create bar, pie, and smith charts; linear, log (semi-log and log-log), and polar plots; axes and grids with tick marks and labels, markers, line types, curves, arcs, circles, ellipses, and more.

The window function allows you to use up to 32 windows. You can set characteristics (such as sizing, scaling, labeling, etc.) for each window individually or you can set identical characteristics for all windows.



INGRAF is available for both FORTRAN-77 and C language (Basic and Pascal available soon).

We also have **GRAFLIB 4.00** for video and printer graphics; **PLOTHI 4.00** and **PLOTHP 4.00** for plotter graphics, and **FORTLIB 4.00**, a FORTRAN enhancement library.

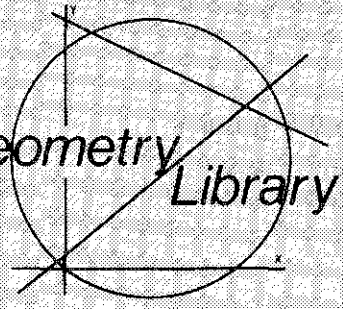
Full source code; no royalties or run-time fees.

Sutrasoft **(713) 491-2088**
P.O. Box 1733 • Sugar Land, TX • 77487

Number **253** on the Reader Service Card

*"The cost involved, in writing just one **TurboGeometry** routine, is more than the price of the library itself".*

Are you programming or planning to program in CAD/CAM or graphics applications? Many hours, even days can be spent in writing and debugging geometric routines. ***Turbo-Geometry library***



can relieve you of these time consuming tasks. There are over 150 routines in the library. Some of the areas covered are: Equations of lines, circles, areas and planes • Intersection of lines, circles, areas and planes • 2 and 3 dimensional transforms • Decomposition of polygon • Hidden line removal • Volume and area calculations • Perspectives • Polygon line clipping • And many more! Supported by example programs, and a 400 page manual. *Source code included. 30 day guarantee. Need* IBM PC/COMP, MS/PC DOS 2.0+. Turbo Geometry is available in Turbo C, Turbo Pascal 4.0, MS C or Turbo Pascal MAC $149.95 plus $5.00 S&H. Foreign orders add $11.00 for S&H. VISA, MC, Chk., MO. To order, call or write.

Disk Software, Inc. • 2116 E. Arapaho, #487
Richardson, TX 75081 • Tel #214-423-7288

"Give yourself a break. Get TurboGeometry Today"

Number **229** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

While it would be more modular to have **EVGADot** restore the state of the Bit Mask register after drawing each dot, it would also be considerably slower to do so. The same could be said of having **EVGADot** set the Enable Set/Reset and Set/Reset registers for each dot: While modularity would improve, speed would suffer markedly.

Drawing Each Line

The **Octant0** and **Octant1** functions draw lines for which |*DeltaX*| is greater than *DeltaY* and lines for which |*DeltaX*| is less than or equal to *DeltaY*, respectively. The parameters to **Octant0** and **Octant1** are the starting point of the line, the length of the line in each dimension, and *XDirection*, the amount by which the *x* coordinate should be changed when it moves. *XDirection* must be either 1 (to draw toward the right edge of the screen) or -1 (to draw toward the left edge of the screen). No value is required for the amount by which the *y* coordinate should be changed: Since *DeltaY* is guaranteed to be positive, the *y* coordinate always changes by 1 dot.

Octant0 draws lines for which |*DeltaX*| is greater than *DeltaY*. For such lines the *x* coordinate of each dot drawn differs from the preceding dot by either 1 or -1, depending on the value of *XDirection*. (This makes it possible for **Octant0** to draw lines in both octant 0 and octant 3.) Whenever *ErrorTerm* becomes nonnegative, indicating that the next *y* coordinate is a better approximation of the line being drawn, the *y* coordinate is increased by 1.

Octant1 draws lines for which |*DeltaX*| is less than or equal to *DeltaY*. For these lines the *y* coordinate of each dot drawn is 1 greater than the *y* coordinate of the previous dot. Whenever *ErrorTerm* becomes nonnegative, indicating that the next *x* coordinate is a better approximation of the line being drawn, the *x* coordinate is advanced by either 1 or -1, depending on the value of *XDirection*. (This makes it possible for **Octant1** to draw lines in both octant 1 and octant 2.)

Drawing Each Dot

At the core of **Octant0** and **Octant1** is a dot-drawing function, **EVGADot**. This function draws a dot at the specified coordinates in whatever color the hardware of the EVGA happens to be set up for. As described earlier, since the entire line drawn by **EVGALine** is of the same color, line-drawing performance is improved by setting up the EVGA's hardware in **EVGALine** once, before the line is drawn, and then drawing all the dots in the line in the same color via **EVGADot**.

EVGADot makes certain assumptions about the screen. First, it assumes that the address of the byte controlling the dots at the start of a given row on the screen is 80 bytes after the start of the row immediately above it; in other words, this implementation of **EVGADot** works only for screens configured to be 80 bytes wide. Since this is the standard configuration of all the modes **EVGALine** is designed to work in, the assumption of 80 bytes per row should be no problem. If it is a problem, however, **EVGADot** could easily be modified to retrieve the BIOS integer variable at address 0040:004A, which contains the number of bytes per row for the current video mode.

Second, **EVGADot** assumes that screen memory is organized as a linear bit map starting at address A000:0000, with the dot at the upper left of the screen controlled by bit 7 of the byte at offset 0, the next dot to the right controlled by bit 6, the ninth dot controlled by bit 7 of the byte at offset 1, and so on. Further, it assumes that the graphics adapter's hardware is configured so that setting the Bit Mask register to allow modification of only the bit controlling the dot of interest and then **OR**ing a value of 0FFH with display memory will draw that dot correctly without affecting any other dots. Again, this is the normal way in which modes 0EH, 0FH, 10H, and 12H operate. **EVGADot** also assumes that the EVGA is set up so that each dot will be drawn in the correct color, as described earlier.

Given those assumptions, **EVGADot** becomes a surprisingly simple function. First, **EVGADot** builds a far pointer that points to the byte of display memory controlling the dot to be drawn. Second, a mask is generated consisting of 0s for all bits except the bit controlling the dot to be drawn. Third, the Bit Mask register is set to that mask, so that when display memory is read and then written, all bits except the one that controls the dot to be drawn will be left unmodified.

Finally, 0FFH is **OR**ed with the display memory byte controlling the dot to be drawn. **OR**ing with 0FFH first reads display memory, thereby loading the EVGA's internal latches with the contents of the display-memory byte controlling the dot to be drawn and then writes display memory with the value 0FFH. Because of the unusual way in which the EVGA's data paths work and the way in which **EVGALine** sets up the EVGA's Enable Set/Reset and Set/Reset registers, the value of 0FFH that is written by the program is ignored. Instead, the value that actually gets placed in display memory is the color that was passed to **EVGALine** and placed in the Set/Reset register. The Bit Mask register (set as described in the preceding paragraph) allows only the single bit controlling the dot to be drawn to be set to this color value.

The result of all this is simply a single dot drawn in the color set up in **EVGALine**. Although **EVGADot** may seem excessively complex for a function that does nothing more than draw one dot, programming the EVGA isn't trivial (as we've seen in the course of many installments of "On Graphics"). Besides, while the explanation of **EVGADot** is lengthy, the code itself is only five lines long.

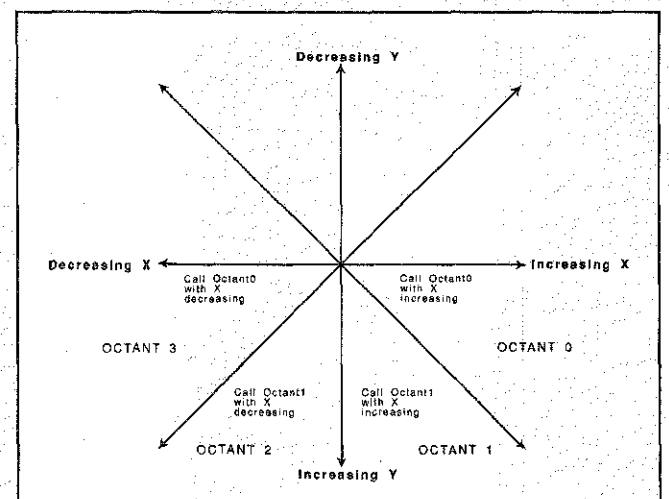


Figure 5. How EVGALine determines which line-drawing function to call and with what direction of change for the *x* coordinate. Lines start at the center of the figure and radiate outward.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

epsilon
programmer's editor

State of the Art editor for DOS, OS/2, Xenix & Unix for PC/XT/AT's

Version 4.0 New!

1988 BYTE AWARD OF DISTINCTION

". . . these products–every one of them–are ones we wouldn't hesitate for a moment to recommend." (Byte, Jan. 1989)

Unrivaled Undo AND Redo

Epsilon gives you the most complete Undo and Redo of any PC programmer's editor. You can undo any sequence of changes with a touch of the Undo key. In fact, you can undo the last few hundred or so operations–you set the limit. And unlike the "full" undo offered by some other editors, Epsilon has Redo, so you can undo your undo's. You can even redo your undo's, and undo your redo's! This flexibility keeps you from losing your work by undoing too much.

Ultimate Customizability . . .

Epsilon features the most powerful extension language of any PC editor. We call it EEL, for Epsilon Extension Language. EEL gives you the rich syntax and data types of C, so you can easily extend or customize the editor to suite your taste. EEL is a real language, not an afterthought. In fact, all of Epsilon's commands are written in EEL, and you get complete source code to all the commands!

". . .but the pièce de résistance is EEL, a C-like extension language that comes with its own compiler. Lugaru Software has implemented Epsilon's entire user-interface in EEL so you can, for example, create your own syntax-sensitive modes or program the help system to browse through your source libraries." (Byte, Jan. 1989)

- C-like extension language, called EEL
- Source code to all commands
- Convenient multilevel undo plus total redo
- Scans compiler errors (customizable)
- Swaps out of memory for big compilers
- Unlimited file size, number of files
- On-line tutorial for quick startup
- As many windows as will fit on the screen
- Supports large displays (like VGA & EGA)
- Context sensitive help autoconfigures
- Tags package indexes subroutine definitions
- Files can be larger than available memory
- Fully utilizes EMS memory
- Interactively customizable keyboard
- EMACS-style command set
- Record keystroke sequences
- Multitasks compilers, even under DOS
- Language support (C, Lisp, etc.)
- Completion on file names, commands, tags, ...
- Full command line editing
- Optional backups
- Regular expression search and tagged replace
- Source level debugger & profiler for EEL
- Do interrupts from EEL (DOS)
- Call any dynamic-link library for EEL (OS/2)
- Saves deleted text (n times)
- Navigate your directory structure
- Not copy protected, for your convenience

. . . & True Multitasking Compiles!

Epsilon's exclusive multitasking facility lets you run your compilers, utilities, or even the DOS command processor, in an Epsilon window, WHILE YOU EDIT. You'll be amazed to see your compiler chugging away while you continue to edit your files. No more dead compile time! If any compiler errors appear, just touch a key and Epsilon parses the error message, then jumps right to the offending line–while your compiler is still running! Don't be fooled by misleading claims from other vendors: Epsilon is the ONLY editor for DOS with this feature! Of course, Epsilon's multitasking also works under OS/2, Xenix, and Unix.

"This feature is especially useful in a programming environment because compilers also can be run in the process buffer. . . If the screen is split into multiple windows, one of which is the process window, the user can see the compilation output while continuing to edit... A single command takes the user to the next error line in a source file and displays the error on the bottom of the screen." (PC Tech Journal, May 1987)

Ask about our free trial!

$195 direct — 60-day money-back guarantee!

Order Hotline: (412) 421-5911

FAX (412) 421-6371

Lugaru Software Ltd. / 5843 Forbes Avenue / Pittsburgh, PA 15217

Number 271 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Line drawing would be somewhat faster if the code of **EVGADot** were made an in-line part of **Octant0** and **Octant1**, thereby saving the overhead of preparing parameters and calling the function. Feel free to do this if you wish. I maintained **EVGADot** as a separate function for clarity and for ease of inserting a dot-drawing function for a different graphics adapter, should that be desired. If you do install a dot-drawing function for a different adapter, remember to remove the EVGA-specific **outportb** lines in **EVGALine** itself.

Comments on the Turbo C Implementation

EVGALine does no error checking whatsoever. My assumption in writing **EVGALine** was that it would ultimately be used as the lowest-level primitive of a graphics software package, with operations such as error checking and clipping performed at a higher level. Similarly, **EVGALine** is tied to the EVGA's screen-coordinate system of (0,0) to (639,349) (in modes 0EH, 0FH, and 10H) or (639,479) (in mode 12H), with the upper left corner considered to be (0,0). Again, transformation from any coordinate system to the coordinate system used by **EVGALine** can be performed at a higher level. **EVGALine** is specifically designed to do one thing: draw lines into the display memory of the EVGA. Additional operations can be supplied by the code that calls **EVGALine**.

The version of **EVGALine** shown in Listing 1 is reasonably fast, but not so fast as it might be. Inclusion of **EVGADot** directly into **Octant0** and **Octant1** and, indeed, inclusion of **Octant0** and **Octant1** directly into **EVGALine** would speed execution by saving the overhead of calling and parameter passing. Handpicked register variables might also speed performance, as would the use of word **outs** (**outport**) rather than byte **outs** (**outportb**). A more significant performance increase would come from eliminating separate calculation of the address and mask for each dot. Since the location of each dot relative to the previous dot is known, the address and mask could be adjusted from one dot to the next rather than recalculated from scratch.

These enhancements are not incorporated into the code in Listing 1, for a couple of reasons. One reason is that it's important that the workings of the algorithm be clearly visible in the code for learning purposes; once the implementation is understood, rewriting it for improved performance would certainly be a worthwhile exercise. Another reason is that when flat-out speed is needed, assembly language is the best way to go—why produce hard-to-understand C code to boost speed a bit when assembly language code can perform the same task at two or more times the speed?

A high-speed assembly language version of **EVGALine** would seem to be a logical next step.

An Assembly Language Implementation

Listing 3 is a high-performance implementation of Bresenham's algorithm written in assembly language. The code is callable from Turbo C—just as Listing 1 is—with the same name (**EVGALine**) and with the same parameters. Either of the two can be linked to any program that calls **EVGALine**, since they appear to be identical to the calling program. The only difference between the two versions is that the sample program in Listing 2 runs more than six times as fast on a PC when calling the assembly language version of **EVGALine** as when calling the Turbo C version and more than four times as fast on an AT. Link each version with Listing 2 and compare performance—the difference is startling.

An explanation of the workings of the code in Listing 3 would be a lengthy one and would be redundant. The basic operation of the code in Listing 3 is no different from that of the code in Listing 1, although the implementation is much changed owing to the nature of assembly language and to its being designed for speed rather than clarity. Given that you thoroughly understand the C implementation in Listing 1, the assembler implementation in Listing 3, being well-commented, should speak for itself.

Listing 3 incorporates a clever notion for which credit is due to Jim Mackraz, who described the notion in a letter written in response to an article I wrote in *PJ*. Jim's suggestion was that when drawing lines for which |*DeltaX*| is greater than |*DeltaY*|, bits set to 1 for each of the dots controlled by a given byte can be accumulated in a register. Rather than each dot being drawn individually, all the dots controlled by that byte can then be drawn at once, with a single access to display memory, when all dot processing associated with that byte has been completed. This approach can save many display-memory reads and writes when drawing nearly horizontal lines, and that's important, since EGAs and VGAs hold the CPU up for a considerable period of time on each display-memory access.

Conclusion

All too many PC programmers fall into the high-level-language trap of thinking that a good algorithm guarantees good performance. Not so. As our two implementations of Bresenham's algorithm graphically illustrate (pun not originally intended, but allowed to stand once recognized), truly good PC code requires both a good algorithm *and* a good assembler implementation. In Listing 3, we've got both—and my, oh my, isn't it fun?

There are several other matters I'd like to attend to, but I've long since overrun my allotted space. In particular, much as I want to get to John Bridges' mode-set code for the 360x480 256-color mode of the VGA, it'll just have to wait. Let me know if there's anything else you'd like to see covered, and I'll try to get to it.

See you next time.

PJ

REFERENCES

Foley, J. D., and Van Dam, A. *Fundamentals of Interactive Computer Graphics*. Reading, Mass.: Addison-Wesley, 1982.

Newman, W. M., and Sproull, R. F. *Principles of Interactive Computer Graphics*. New York: McGraw-Hill, 1979.

Michael Abrash is a Senior Software Engineer for Orion Instruments, Inc., of Redwood City, California, where he is working on the design of advanced PC-based test instrumentation. Readers may write to Michael at P.O. Box 39051, Mountain View, CA 94039, or drop him a line via MCI Mail, user name MABRASH.

Code follows.

PRODUCED 2002 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Proven Compilers Get Solid Results!!!

The fastest Fortran 77/386 Compiler Around

- Compiles 13,000 lpm on a 386
- Runs under MS-DOS with DOS extenders
- Optional checkout mode with run-time diagnostics
- Full window-based source level debugger
- Dynamically allows linking to libraries during program's execution
- Contains a full 32-bit in-line symbolic assembler with window-based debugger
- Addresses up to 4 Gigabytes of memory

OTG SYSTEMS, INC.™

Distributed by OTG Systems, Inc.

Our compilers run faster, do more, cost less and increase system and people productivity!

Suite 300, 308 Mulberry Street, Scranton, Pennsylvania 18505-5250
(717) 343-8200

MS-DOS is a trademark of Microsoft Corporation. FTN77/386 is developed by the UNIVERSITY OF SALFORD

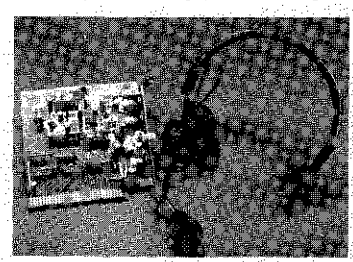
Number **270** on the Reader Service Card

VOICE MASTER KEY® VOICE RECOGNITION SYSTEM

FOR PC/COMPATIBLES & TANDY 1000 SERIES
A FULL FEATURED VOICE I/O SYSTEM

GIVE A NEW DIMENSION TO PERSONAL COMPUTING. . .The amazing **Voice Master Key System** adds recognition to just about any program or application. Voice command up to 256 keyboard macros from within CAD, desktop publishing, word processing, spread sheet, or game programs. Fully TSR and occupies less than 64K. Instant response time and high recognition accuracy. Voice recognition tool-box utilities are included. **A genuine productivity enhancer!**

SPEECH RECORDING SOFTWARE. . .Digitally record your own speech, sound, or music to put into your own software programs. Software provides sampling rate variations, graphics-based editing, and data compression utilities. Create software sound files you can add to macros for voice recognition verification response. **a complete, superior speech and sound development tool.**



SOFTWARE CONVERSION CODES. . .The **Voice Master Key System** operates a growing list of third party talking software titles using synthesized phonetics (text - to - speech) or digitized PCM, ADPCM, and CVSDM encoded sound files. **Voice Master Key system does it all!**

EVERYTHING INCLUDED. **Voice Master Key System** consists of a plug-in card, durable lightweight microphone headset, software, and manual. Card fits any available slot. External ports consist of mic inputs and volume controlled output sockets. **High quality throughout, easy and fun to use.**

ONLY $149.95 COMPLETE

ONLY $89.95 FOR TANDY 1000 SL/TL MODELS SOFTWARE ONLY—REQUIRES TANDY BRAND ELECTRET MICROPHONE.

ORDER HOTLINE: (503) 342-1271 Monday-Friday, 8AM to 5 PM Pacific Time

Visa/MasterCard, company checks, money orders, CODs (with prior approval) accepted. Personal checks subject to 3 week shipping delay. Specify computer type and disk format (3½" or 5¼") when ordering. Add $5 shipping charge for delivery in USA and Canada. Foreign inquiries contact Covox for C & F quotes. 30 DAY MONEY BACK GUARANTEE IF NOT COMPLETELY SATISFIED. ONE YEAR WARRANTY ON HARDWARE.

CALL OR WRITE FOR **FREE** PRODUCT CATALOG

COVOX INC.
675 Conger Street, Eugene, OR 97402
TEL: 503-342-1271 • FAX: 503-342-1283

Number **277** on the Reader Service Card

SCREEN MANAGER P·R·O·F·E·S·S·I·O·N·A·L

ADVANCED WINDOWING

• The Screen Manager Professional gives your applications the finest bullet proof windowing interface available. All functions have been carefully designed and tested with the programmer in mind. The functions are easy to use and remember. Although other products may boast of hundreds of functions, they often involve needless repetition. Our functions are elegant and powerful. Each accomplishes a variety of tasks with one command. All functions are written in hand-optimized assembly language to provide the fastest possible response and smallest possible size.

ORDER

• To receive your copy of the **Screen Manager Professional, send $139.95.** VISA, MasterCard, Check, or Money Order accepted. Sorry, no C.O.D.'s. Please allow 2-4 weeks for delivery. PA residents add 6% sales tax.

IMPORTANT FEATURES

- *Unlimited number of windows*
- Complete writing and scrolling support for overlapping windows
- VGA, EGA, CGA, MGA support
- Write to video ram or use BIOS
- Multiple monitor support
- No fuzz or flicker
- Drag windows around the screen
- Menu system
- Resize and reposition windows
- Save and Load windows from disk
- MS-DOS memory allocation
- 100% hand-optimized assembly
- Over 90 powerful functions
- Royalty free!

FOR USE WITH

- Datalight C • Microsoft C
- Microsoft Quick C • Aztec C • Turbo C
- Microsoft Quickbasic • Turbo Pascal

EASY TO USE

• Free yourself to concentrate on the substance of your program. Over 90 functions provide all the windowing tools needed to develop serious applications whether you are a professional or a hobbyist.

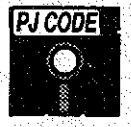
• The Screen Manager Professional is a powerful windowing facility. You can increase your productivity as well as the overall quality of your work. The most important part of any program is its ease of use. The Screen Manager Professional can help you create easy to use, visually stimulating, programs with a minimum of effort. Now powerful windowing is available to you at an affordable price. If you have further questions, please call or write.

Programming Utilities for Professionals

LAI
LOGICAL ALTERNATIVES INCORPORATED
Calder Square
P.O. Box 10674
State College, PA 16805-0674
(814) 234-8088

Number **246** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "Fast Line Drawing . . ." by Michael Abrash

1

LISTING 1

```
/* Turbo C implementation of Bresenham's line drawing algorithm for the EGA
 * and VGA. Works in modes 0xE, 0xF, 0x10, and 0x12. Compiled with Turbo C 2.0.
 * By Michael Abrash.  2/4/89.
 */
#include <dos.h>      /* contains MK_FP macro */
#define EVGA_SCREEN_WIDTH_IN_BYTES   80  /* memory offset from start of
                                            one row to start of next */
#define EVGA_SCREEN_SEGMENT          0xA000  /* display memory segment */
#define GC_INDEX                     0x3CE   /* Graphics Controller Index register port */
#define GC_DATA                      0x3CF   /* Graphics Controller Data register port */
#define SET_RESET_INDEX              0  /* indexes of needed */
#define ENABLE_SET_RESET_INDEX       1  /* Graphics Controller */
#define BIT_MASK_INDEX               8  /* registers */

/* Draws a dot at (X0,Y0) in whatever color the EGA/VGA hardware is
 * set up for. Leaves the bit mask set to whatever value the dot required.
 */
void EVGADot(X0, Y0)
unsigned int X0;      /* coordinates at which to draw dot, with */
unsigned int Y0;      /* (0,0) at the upper left of the screen */
{
   unsigned char far *PixelBytePtr;
   unsigned char PixelMask;

   /* Calculate the offset in the screen segment of the byte in which the pixel lies */
   PixelBytePtr = MK_FP(EVGA_SCREEN_SEGMENT,
      ( Y0 * EVGA_SCREEN_WIDTH_IN_BYTES ) + ( X0 / 8 ));

   /* Generate a mask with a 1 bit in the pixel's position within the screen byte */
   PixelMask = 0x80 >> ( X0 & 0x07 );

   /* Set up the Graphics Controller's Bit Mask register to allow only the bit
      corresponding to the pixel being drawn to be modified */
   outportb(GC_INDEX, BIT_MASK_INDEX);
   outportb(GC_DATA, PixelMask);

   /* Draw the pixel. Because of the operation of the set/reset feature of the EGA/VGA,
      the value written doesn't matter. The screen byte is ORed in order to perform
      a read to latch the display memory, then perform a write in order to modify it. */
   *PixelBytePtr |= 0xFF;
}

/* Draws a line in octant 0 or 3 ( |DeltaX| >= DeltaY ).
 * |DeltaX|+1 points are drawn.
 */
void Octant0(X0, Y0, DeltaX, DeltaY, XDirection)
unsigned int X0, Y0;          /* coordinates of start of the line */
unsigned int DeltaX, DeltaY;  /* length of the line */
int XDirection;               /* 1 if line is drawn left to right,
                                 -1 if drawn right to left */
{
   int DeltaYx2;
   int DeltaYx2MinusDeltaXx2;
   int ErrorTerm;

   /* Set up initial error term and values used inside drawing loop */
   DeltaYx2 = DeltaY * 2;
   DeltaYx2MinusDeltaXx2 = DeltaYx2 - (int) ( DeltaX * 2 );
   ErrorTerm = DeltaYx2 - (int) DeltaX;

   /* Draw the line */
   EVGADot(X0, Y0);                  /* draw the first pixel */
   while ( DeltaX-- ) {
      /* See if it's time to advance the Y coordinate */
      if ( ErrorTerm >= 0 ) {
         /* Advance the Y coordinate & adjust the error term back down */
```

2

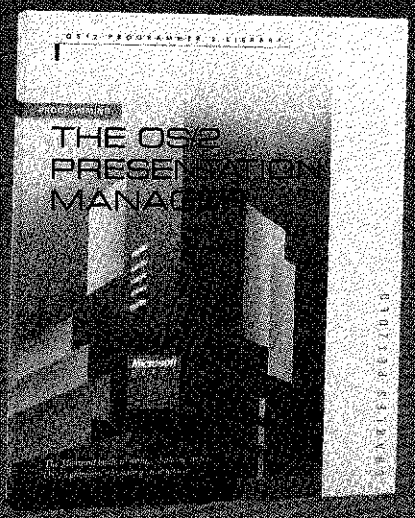
```
         Y0++;
         ErrorTerm += DeltaYx2MinusDeltaXx2;
      } else {
         ErrorTerm += DeltaYx2;  /* Add to the error term */
      }
      X0 += XDirection;          /* advance the X coordinate */
      EVGADot(X0, Y0);           /* draw a pixel */
   }
}

/* Draws a line in octant 1 or 2 ( |DeltaX| < DeltaY ).
 * |DeltaY|+1 points are drawn.
 */
void Octant1(X0, Y0, DeltaX, DeltaY, XDirection)
unsigned int X0, Y0;          /* coordinates of start of the line */
unsigned int DeltaX, DeltaY;  /* length of the line */
int XDirection;               /* 1 if line is drawn left to right,
                                 -1 if drawn right to left */
{
   int DeltaXx2;
   int DeltaXx2MinusDeltaYx2;
   int ErrorTerm;

   /* Set up initial error term and values used inside drawing loop */
   DeltaXx2 = DeltaX * 2;
   DeltaXx2MinusDeltaYx2 = DeltaXx2 - (int) ( DeltaY * 2 );
   ErrorTerm = DeltaXx2 - (int) DeltaY;

   EVGADot(X0, Y0);              /* draw the first pixel */
   while ( DeltaY-- ) {
      /* See if it's time to advance the X coordinate */
      if ( ErrorTerm >= 0 ) {
         /* Advance the X coordinate & adjust the error term back down */
         X0 += XDirection;
         ErrorTerm += DeltaXx2MinusDeltaYx2;
      } else {
         ErrorTerm += DeltaXx2;    /* Add to the error term */
      }
      Y0++;                        /* advance the Y coordinate */
      EVGADot(X0, Y0);             /* draw a pixel */
   }
}

/* Draws a line on the EGA or VGA. */
void EVGALine(X0, Y0, X1, Y1, Color)
int X0, Y0;    /* coordinates of one end of the line */
int X1, Y1;    /* coordinates of the other end of the line */
char Color;    /* color to draw line in */
{
   int DeltaX, DeltaY;
   int Temp;

   /* Set the drawing color and put the drawing color in the Set/Reset register */
   outportb(GC_INDEX, SET_RESET_INDEX);
   outportb(GC_DATA, Color);
   /* Cause all planes to be forced to the Set/Reset color */
   outportb(GC_INDEX, ENABLE_SET_RESET_INDEX);
   outportb(GC_DATA, 0xF);

   /* Save half the line-drawing cases by swapping Y0 with Y1
      and X0 with X1 if Y0 is greater than Y1. As a result, DeltaY
      is always > 0, and only the octant 0-3 cases need to be handled. */
   if ( Y0 > Y1 ) {
      Temp = Y0;
      Y0 = Y1;
      Y1 = Temp;
      Temp = X0;
      X0 = X1;
      X1 = Temp;
   }

   /* Handle as four separate cases, the four octants in which Y1 is greater than Y0 */
   DeltaX = X1 - X0;    /* calculate the length of the line in each coordinate */
   DeltaY = Y1 - Y0;
```

Code continues . . .

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

OS/2 and Presentation Manager...

When You Need Reliable Information, Go to the Source.



PROGRAMMING THE OS/2 PRESENTATION MANAGER

Charles Petzold

Here is the first full-length book on the long-awaited programming API for OS/2 1.1 and 1.2 Presentation Manager. If you're developing OS/2 applications, this book will guide you through Presentation Manager's new features. Packed with practical examples, it covers all the basics and includes dozens of valuable C programs and routines. Learn how to: ■ Create overlapping windows ■ Use the mouse and keyboard ■ Control window size and position ■ Draw in windows ■ Manipulate input and output ■ Control child windows ■ Use bitmaps, icons, metafiles, and strings ■ Customize the mouse and keyboard accelerators ■ Explore with device units ■ Understand dynamic linking ■ And more. An encompassing title resource. $24.95

Order Code 86-96147

ADVANCED OS/2 PROGRAMMING

Ray Duncan

Authoritative information, expert advice, and great examples in assembly language and C make this comprehensive overview of OS/2 indispensable to anyone programming the OS/2 kernel. Duncan addresses a broad range of significant OS/2 issues, programming the user interface, mass storage, memory management, multitasking, interprocess communications, customizing filters, device drivers, and monitors, and using OS/2 dynamic link libraries. A valuable reference section includes detailed information on each of the more than 250 system service calls in version 1.1 of the OS/2 kernel. $24.95

Order Code 86-96106

Microsoft® Press

Hardcore Computer Books

Available wherever books and software are sold.

Or place your credit card order by calling **1-800-638-3030** *(8AM to 4PM EST). In MD call 824-7300.*

Number 234 on the Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

3

```
    if ( DeltaX > 0 ) {
        if ( DeltaX > DeltaY )      Octant0(X0, Y0, DeltaX, DeltaY, 1);
        else                        Octant1(X0, Y0, DeltaX, DeltaY, 1);
    } else {
        DeltaX = -DeltaX;           /* absolute value of DeltaX */
        if ( DeltaX > DeltaY )      Octant0(X0, Y0, DeltaX, DeltaY, -1);
        else                        Octant1(X0, Y0, DeltaX, DeltaY, -1);
    }

    /* Return the state of the EGA/VGA to normal */
    outportb(GC_INDEX, ENABLE_SET_RESET_INDEX);
    outportb(GC_DATA, 0);
    outportb(GC_INDEX, BIT_MASK_INDEX);
    outportb(GC_DATA, 0xFF);
}
```

LISTING 2

```
/* Sample program to illustrate EGA/VGA line drawing routines. Compiled with Turbo C 2.0.
 *
 * By Michael Abrash.  2/4/89.
 */
#include <dos.h>        /* contains geninterrupt */

#define GRAPHICS_MODE   0x10
#define TEXT_MODE       0x03
#define BIOS_VIDEO_INT  0x10
#define X_MAX           640     /* working screen width */
#define Y_MAX           348     /* working screen height */

extern void EVGALine();

/* Subroutine to draw a rectangle full of vectors, of the specified
 * length and color, around the specified rectangle center.
 */
void VectorsUp(XCenter, YCenter, XLength, YLength, Color)
int XCenter, YCenter;   /* center of rectangle to fill */
int XLength, YLength;   /* distance from center to edge of rectangle */
int Color;              /* color to draw lines in */
{
    int WorkingX, WorkingY;

    /* Lines from center to top of rectangle */
    WorkingX = XCenter - XLength;
    WorkingY = YCenter - YLength;
    for ( ; WorkingX < ( XCenter + XLength ); WorkingX++ )
        EVGALine(XCenter, YCenter, WorkingX, WorkingY, Color);

    /* Lines from center to right of rectangle */
    WorkingX = XCenter + XLength - 1;
    WorkingY = YCenter - YLength;
    for ( ; WorkingY < ( YCenter + YLength ); WorkingY++ )
        EVGALine(XCenter, YCenter, WorkingX, WorkingY, Color);

    /* Lines from center to bottom of rectangle */
    WorkingX = XCenter + XLength - 1;
    WorkingY = YCenter + YLength - 1;
    for ( ; WorkingX >= ( XCenter - XLength ); WorkingX-- )
        EVGALine(XCenter, YCenter, WorkingX, WorkingY, Color);

    /* Lines from center to left of rectangle */
    WorkingX = XCenter - XLength;
    WorkingY = YCenter + YLength - 1;
    for ( ; WorkingY >= ( YCenter - YLength ); WorkingY-- )
        EVGALine(XCenter, YCenter, WorkingX, WorkingY, Color );
}

/* Sample program to draw four rectangles full of lines. */
int main()
{
    char temp;

    /* Set graphics mode */
    _AX = GRAPHICS_MODE;
```

4

```
    geninterrupt(BIOS_VIDEO_INT);

    /* Draw each of four rectangles full of vectors */
    VectorsUp(X_MAX / 4, Y_MAX / 4, X_MAX / 4, Y_MAX / 4, 1);
    VectorsUp(X_MAX * 3 / 4, Y_MAX / 4, X_MAX / 4, Y_MAX / 4, 2);
    VectorsUp(X_MAX / 4, Y_MAX * 3 / 4, X_MAX / 4, Y_MAX / 4, 3);
    VectorsUp(X_MAX * 3 / 4, Y_MAX * 3 / 4, X_MAX / 4, Y_MAX / 4, 4);

    /* Wait for the enter key to be pressed */
    scanf("%c", &temp);

    /* Back to text mode */
    _AX = TEXT_MODE;
    geninterrupt(BIOS_VIDEO_INT);
}
```

LISTING 3

```
; Fast assembler implementation of Bresenhem's line drawing algorithm for the EGA and VGA.
; Works in modes 0Dh, 0Eh, 0Fh, 10h, and 12h. Turbo C version 2.0 near-callable.
; Bit mask accumulation technique when |DeltaX| >= |DeltaY|  suggested by Jim Mackraz.
; Assembled with TASM 1.0.
;
; By Michael Abrash.  2/4/89.
;
;*****************************************************************
; C-compatible line-drawing entry point at _EVGALine.            *
; Called from Turbo C with:  EVGALine(X0, Y0, X1, Y1, Color);    *
;*****************************************************************

_TEXT   segment byte public 'CODE'
_TEXT   ends
CONST   segment word public 'CONST'
CONST   ends
_BSS    segment word public 'BSS'
_BSS    ends
_DATA   segment word public 'DATA'
_DATA   ends
DGROUP  group   CONST, _BSS, _DATA
        assume  cs: _TEXT, ds: DGROUP, ss: DGROUP, es: DGROUP
_TEXT   segment

; Equates.
EVGA_SCREEN_WIDTH_IN_BYTES equ  80      ;memory offset from start of one row
                                        ; to start of next in display memory
EVGA_SCREEN_SEGMENT     equ     0a000h  ;display memory segment.
GC_INDEX                equ     3ceh    ;Graphics Controller Index register port
SET_RESET_INDEX         equ     0       ;indexes of needed
ENABLE_SET_RESET_INDEX  equ     1       ; Graphics Controller
BIT_MASK_INDEX          equ     8       ; registers

; Stack frame.
EVGALineParms   struc
        dw      ?               ;pushed BP
        dw      ?               ;pushed return address (make double word for far call)
X0      dw      ?               ;starting X coordinate of line
Y0      dw      ?               ;starting Y coordinate of line
X1      dw      ?               ;ending X coordinate of line
Y1      dw      ?               ;ending Y coordinate of line
Color   db      ?               ;color of line
        db      ?               ;dummy to pad to word size
EVGALineParms   ends

;*****************************************************************
; Line drawing macros.                                           *
;*****************************************************************

; Macro to loop through length of line, drawing each pixel in turn.
; Used for case of |DeltaX| >= |DeltaY|. |DeltaX|+1 points are drawn.
;
; Input:
;       MOVE_LEFT: 1 if DeltaX < 0, 0 else
;       AL: pixel mask for initial pixel
;       BX: |DeltaX| (X distance between start and end points)
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED.

5

```
;       DX: address of GC data register, with index register set to
;           index of Bit Mask register
;       SI: DeltaY (Y distance between start and end points)
;       ES:DI: display memory address of byte containing initial pixel
; Output: none
;
LINE1   macro   MOVE_LEFT
        local   LineLoop, MoveXCoord, NextPixel, Line1End
        local   MoveToNextByte, ResetBitMaskAccumulator
        mov     cx,bx   ;# of pixels in line
        jcxz    Line1End ;done if there are no more pixels (there's always at
                         ; least the one pixel at the start location)
        shl     si,1    ;DeltaY * 2
        mov     bp,si   ;error term
        sub     bp,bx   ;error term starts at DeltaY * 2 - DeltaX
        shl     bx,1    ;DeltaX * 2
        sub     si,bx   ;DeltaY * 2 - DeltaX * 2 (used in loop)
        add     bx,si   ;DeltaY * 2 (used in loop)
        mov     ah,al   ;set aside pixel mask for initial pixel with AL (the
                        ; pixel mask accumulator) set for the initial pixel
LineLoop:
; See if it's time to advance the Y coordinate yet.
        and     bp,bp       ;see if error term is negative
        js      MoveXCoord  ;yes, stay at the same Y coordinate
;
; Advance the Y coordinate, first writing all pixels in the current
; byte, then move the pixel mask either left or right, depending
; on MOVE_LEFT.
        out     dx,al   ;set up bit mask for pixels in this byte
        xchg    byte ptr [di],al
                        ;load latches and write pixels, with bit mask preserving other
                        ; latched bits. Because set/reset is enabled for all planes,
                        ; the value written actually doesn't matter
        add     di,SVGA_SCREEN_WIDTH_IN_BYTES ;increment Y coordinate
        add     bp,si   ;adjust error term back down
;
; Move pixel mask one pixel (either right or left, depending
; on MOVE_LEFT), adjusting display memory address when pixel mask wraps.
        if MOVE_LEFT
            rol     ah,1    ;move pixel mask 1 pixel to the left
        else
            ror     ah,1    ;move pixel mask 1 pixel to the right
        endif
            jnc     ResetBitMaskAccumulator ;didn't wrap to next byte
            jmp     short MoveToNextByte    ;did wrap to next byte
;
; Move pixel mask one pixel (either right or left, depending
; on MOVE_LEFT), adjusting display memory address and writing pixels
; in this byte when pixel mask wraps.
MoveXCoord:
        add     bp,bx   ;increment error term & keep same
        if MOVE_LEFT
            rol     ah,1    ;move pixel mask 1 pixel to the left
        else
            ror     ah,1    ;move pixel mask 1 pixel to the right
        endif
            jnc     NextPixel ;if still in same byte, no need to
                              ; modify display memory yet
            out     dx,al   ;set up bit mask for pixels in this byte.
            xchg    byte ptr [di],al
                            ;load latches and write pixels, with bit mask
                            ; preserving other latched bits. Because
                            ; set/reset is enabled for all planes, the
                            ; value written actually doesn't matter
        MoveToNextByte:
        if MOVE_LEFT
            dec     di      ;next pixel is in byte to left
        else
            inc     di      ;next pixel is in byte to right
        endif
        ResetBitMaskAccumulator:
            sub     al,al   ;reset pixel mask accumulator
        NextPixel:
            or      al,ah   ;add the next pixel to the pixel mask accumulator
```

Code continues . . .

PROTECT INTELLIGENCE WITH INTELLIGENCE

PROTECH KEYs:

- ❑ are totally transparent to the end user
- ❑ allow unlimited back up copies
- ❑ free up disk drives
- ❑ do not interfere with peripheral operations
- ❑ are easy to install
- ❑ protect software developers' revenues
- ❑ provide the highest level of protection available

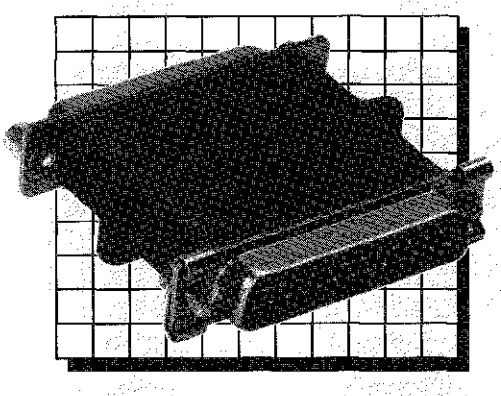
PROTECH'S KEYs provide the highest level of protection because of:

- ❑ assembler based routines (not drivers)
- ❑ encrypted routines
- ❑ physically unique hardware KEYs
- ❑ sophisticated software installation advice

THE KEY

The KEY provides effective software protection while insuring customer satisfaction. The KEY is a random response device which is designed for identically reproduced software packages.

SOFTWARE PROTECTION KEYs BY PROTECH



WE HAVE THE *KEY* TO SOFTWARE COPY PROTECTION

For a demonstration package or additional information, please write or call:

1-800-843-0413

THE MEMORY KEY

The MEMORY KEY is a programmable software protection device. Each byte of memory can be addressed in groups or individually. Possible applications for the MEMORY KEY include:

- ❑ modular package control
- ❑ serialization
- ❑ customization
- ❑ access control to PCs

Information stored in the MEMORY KEY can also be altered by your application software in the field. This dynamic writing to the MEMORY KEY opens up unlimited possibilities. Possible applications for the dynamic read/write function include:

- ❑ demo control
- ❑ software leasing
- ❑ updating modules in the field
- ❑ any "counter" operation

PROTECH MARKETING, INC.
1804 W. Southern Parkway • Building A-112
Durham, North Carolina 27707
(919) 490-4970 FAX (919) 490-4974

Number **228** on the Reader Service Card

PRODUCED 2023 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Commenting Disassembler!

SOURCER

- **SEE HOW PROGRAMS WORK**
- **EASILY MODIFY PROGRAMS**

SOURCER™ creates detailed commented source code and listings from memory and executable files. Built in data analyzer and simulator resolves data across multiple segments and provides detailed comments on interrupts and subfunctions, I/O ports and much more. Determines necessary assembler directives for reassembly. Includes a definition file facility to include your own remarks and descriptive labels, force data types, and more. Complete support for 8088/87 through 80286/287 and V20/V30 instruction sets. We welcome comparisons with any other product, because no product comes close to the ease of use and output clarity of SOURCER.

On my list of programs that I simply won't do without!
—Robert Hummel, Senior Technical Editor, PC Magazine, 4/26/88

SAMPLE OUTPUT

Fully automatic

Program header

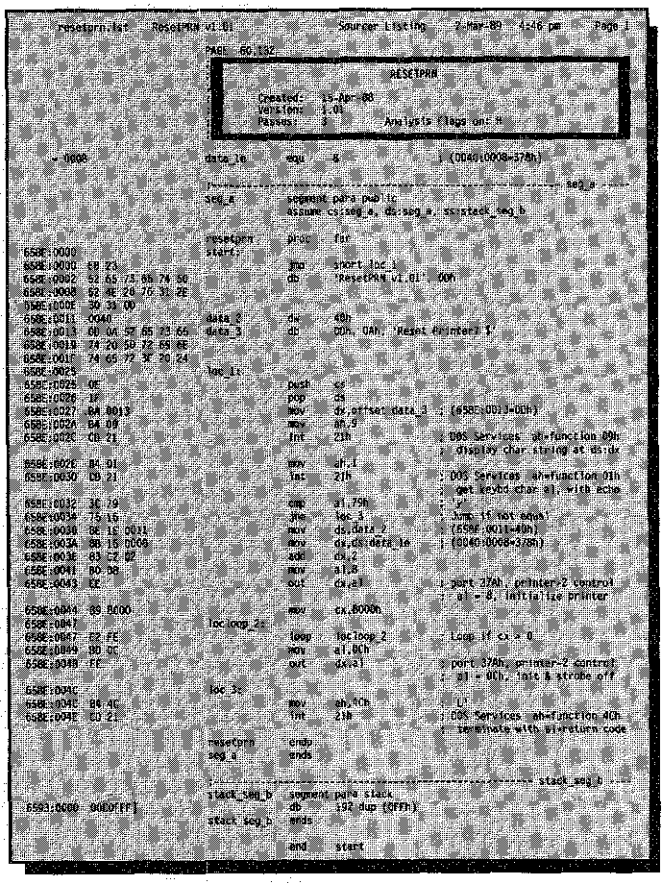
Assembler directives

Determines data areas and type

Detailed comments

Simulator follows segment changes

Easy to read format



(Source code output and inline cross reference can also be selected)

CBOS Source

- **CHANGE AND ADD FEATURES**
- **CLARIFY INTERFACES**

for PS/2, AT, XT, PC, and Clones

The BIOS Pre-Processor™ with SOURCER provides the first means to obtain accurate legal source listings for any BIOS! Identifies entry points with full explanations. Resolves PS/2's multiple jumps for improved clarity. Provides highly descriptive labels such as "video_mode" and much more. Fully automatic.

SOURCER Commenting disassembler	$99.95	**UNPACKER**™ Unpack packed EXE files and more	$39.95
SOURCER with BIOS Pre-Processor	139.95	**ASMtool**™ Assembly source analyzer and flowcharter	89.95

USA Shipping & Handling $3; Outside USA $15; CA Res. add sales tax; PS/2, AT, XT, and PC are trademarks of IBM Corp.

All our products come with a 30 day money back satisfaction guarantee. Not copy protected. To order or receive additional information just call!

VISA **1-800-662-8266** MasterCard

V COMMUNICATIONS, INC.
3031 Tisch Way, Suite 905, Dept. J3, San Jose, CA 95128 (408) 296-4224

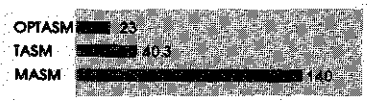
Number **125** on the Reader Service Card

THE PROGRAMMER'S CHOICE

OPTASM
- fastest assembler available
- eliminates phase errors
- never inserts spurious NOPs
- complete TSR online help
- includes a linker and debugger
- "OPTASM is clearly technically superior to MASM"
  Computer Language - Dec 88

$125

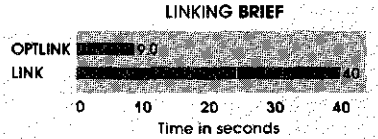
ASSEMBLING BGIDEMO ON PC



OPTLINK
- fastest linker available
- generates EXE, COM or SYS
- generates global XREF
- supports CodeView info
- works with MS C5.1

$125

LINKING BRIEF



OPTLIB
- 10 times faster than LIB
- * and ? in filenames
- time-based replace
- LIB-compatible syntax
- rename PUBLICs and EXTRNs

$49

BUILD 670K .LIB FILE

OPTDEBUG
- source level - supports CV info
- online help system
- reverse execution
- 50k COM file
- debug EXEs, COMs and TSRs

$125

SLR SYSTEMS
1622 N. Main Street Butler, PA 16001
412-282-0864 TELEX 559215

800-833-3061 (USA)
Call today to order!

Number **165** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Abrash code, continued

6

```
        loop    LineLoop
;
; Write the pixels in the final byte.
LinelEnd:
        out     dx,al   ;set up bit mask for pixels in this byte.
        xchg    byte ptr [di],al
                        ;load latches and write pixels, with bit mask
                        ; preserving other latched bits. Because
                        ; set/reset is enabled for all planes, the
                        ; value written actually doesn't matter
        endm

; Macro to loop through length of line, drawing each pixel in turn.
; Used for case of DeltaX < DeltaY. |DeltaY|+1 points are drawn.
;
; Input:
;       MOVE_LEFT: 1 if DeltaX < 0, 0 else
;       AL: pixel mask for initial pixel
;       BX: |DeltaX| (X distance between start and end points)
;       DX: address of GC data register, with index register set to
;               index of Bit Mask register
;       SI: DeltaY (Y distance between start and end points)
;       ES:DI: display memory address of byte containing initial pixel
; Output: none
;
LINE2   macro   MOVE_LEFT
        local   LineLoop, MoveYCoord, ETermAction, Line2End
        mov     cx,si   ;# of pixels in line
        shl     bx,1    ;DeltaX * 2
        mov     bp,bx   ;error term
        sub     bp,si   ;error term starts at DeltaX * 2 - DeltaY
        shl     si,1    ;DeltaY * 2
        sub     bx,si   ;DeltaX * 2 - DeltaY * 2 (used in loop)
        add     si,bx   ;DeltaX * 2 (used in loop)
;
; Set up initial bit mask & draw initial pixel.
        out     dx,al
        xchg    byte ptr [di],ah
                        ;load latches and write pixel, with bit mask
                        ; preserving other latched bits. Because
                        ; set/reset is enabled for all planes, the
                        ; value written actually doesn't matter
        jcxz    Line2End ;done if there are no more pixels (there's always
                        ; at least the one pixel at the start location)
LineLoop:
; See if it's time to advance the X coordinate yet.
;
        and     bp,bp           ;see if error term is negative
        jns     ETermAction     ;no, advance X coordinate
        add     bp,si           ;increment error term & keep same
        jmp     short MoveYCoord ; X coordinate
ETermAction:
; Move pixel mask one pixel (either right or left, depending
; on MOVE_LEFT), adjusting display memory address when pixel mask wraps.
;
if MOVE_LEFT
        rol     al,1
        sbb     di,0
else
        ror     al,1
        adc     di,0
endif
        out     dx,al   ;set new bit mask
        add     bp,bx   ;adjust error term back down
;
; Advance Y coordinate.
MoveYCoord:
        add     di,EVGA_SCREEN_WIDTH_IN_BYTES
;
; Write the next pixel.
        xchg    byte ptr [di],ah
                        ;load latches and write pixel, with bit mask
                        ; preserving other latched bits. Because
                        ; set/reset is enabled for all planes, the
```

7

```
                        ; value written actually doesn't matter
        loop    LineLoop
Line2End:
        endm

;*****************************************************************
; Line drawing routine.                                          *
;*****************************************************************

        public  _EVGALine
_EVGALine       proc    near
        push    bp
        mov     bp,sp
        push    si      ;preserve register variables
        push    di
        push    ds
;
; Point DS to display memory.
        mov     ax,EVGA_SCREEN_SEGMENT
        mov     ds,ax
;
; Set the Set/Reset and Set/Reset Enable registers for the selected color.
        mov     dx,GC_INDEX
        mov     al,SET_RESET_INDEX
        out     dx,al
        inc     dx
        mov     al,[bp+Color]
        out     dx,al
        dec     dx
        mov     al,ENABLE_SET_RESET_INDEX
        out     dx,al
        inc     dx
        mov     al,0ffh
        out     dx,al
;
; Get DeltaY.
        mov     si,[bp+Y1]      ;line Y start
        mov     ax,[bp+Y0]      ;line Y end, used later in calculating start address
        sub     si,ax           ;calculate DeltaY
        jns     CalcStartAddress ;if positive, we're set
;
; DeltaY is negative -- swap coordinates so we're always working
; with a positive DeltaY.
        mov     ax,[bp+Y1]      ;set line start to Y1, for use in calculating start
address
        mov     dx,[bp+X0]
        xchg    dx,[bp+X1]
        mov     [bp+X0],dx      ;swap X coordinates
        neg     si              ;convert to positive DeltaY
;
; Calculate the starting address in display memory of the line.
; Hardwired for a screen width of 80 bytes.
CalcStartAddress:
        shl     ax,1    ;Y0 * 2 ;Y0 is already in AX
        shl     ax,1    ;Y0 * 4
        shl     ax,1    ;Y0 * 8
        shl     ax,1    ;Y0 * 16
        mov     di,ax
        shl     ax,1    ;Y0 * 32
        shl     ax,1    ;Y0 * 64
        add     di,ax   ;Y0 * 80
        mov     dx,[bp+X0]
        mov     cl,dl   ;set aside lower 3 bits of column for
        and     cl,7    ; pixel masking
        shr     dx,1
        shr     dx,1
        shr     dx,1    ;get byte address of column (X0/8)
        add     di,dx   ;offset of line start in display segment
;
; Set up GC Index register to point to the Bit Mask register.
        mov     dx,GC_INDEX
        mov     al,BIT_MASK_INDEX
        out     dx,al
        inc     dx      ;leave DX pointing to the GC Data register
```

Code continues . . .

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Abrash code, continued

```
; Set up pixel mask (in-byte pixel address).
        mov     al,80h
        shr     al,cl
;
; Calculate DeltaX.
        mov     bx,[bp+X1]
        sub     bx,[bp+X0]
;
; Handle correct one of four octants.
        js      NegDeltaX
        cmp     bx,si
        jb      Octant1
;
; DeltaX >= DeltaY >= 0.
        LINE1   0
        jmp     EVGALineDone
;
; DeltaY > DeltaX >= 0.
Octant1:
        LINE2   0
        jmp     short EVGALineDone
;
NegDeltaX:
        neg     bx              ;|DeltaX|
        cmp     bx,si
        jb      Octant2
;
; |DeltaX| >= DeltaY and DeltaX < 0.
        LINE1   1
        jmp     short EVGALineDone
;
; |DeltaX| < DeltaY and DeltaX < 0.
Octant2:
        LINE2   1
;
EVGALineDone:
; Restore EVGA state.
;
        mov     al,0ffh
        out     dx,al   ;set Bit Mask register to 0ffh
        dec     dx
        mov     al,ENABLE_SET_RESET_INDEX
        out     dx,al
        inc     dx
        sub     al,al
        out     dx,al   ;set Enable Set/Reset register to 0
;
        pop     ds
        pop     di
        pop     si
        pop     bp
        ret
_EVGALine       endp
;
_TEXT   ends
        end
```


The Heap Expander™ version 2.5

Now your programs can have virtually unlimited heap space using expanded memory, extended memory, disk space, or any combination of the three. And it's all transparent. The Heap Expander's initialization code checks the system's resources and uses whatever is available.

only $79.95*

- Uses LIM-standard expanded memory if present.
- Uses AT-style extended memory if present.
- Swaps data to disk as needed.

Libraries and Source Code for:

- Turbo C
- Microsoft C 4.0 and 5.0
- Turbo Pascal 3.0 and 4.0

Requires an IBM PC, XT, AT, or close compatible with MS-DOS or PC-DOS version 2.0 or above

MC/VISA/COD call
1-800-248-1045 x 100 (US)
1-800-952-5560 x 100 (Idaho)

*CA residents add 6 1/2% sales tax. Foreign customers, please add $4.00 for shipping and handling.

The Tool Makers

P. O. Box 2151
Santa Cruz, CA 95063
(408) 485-0690

Number 166 on the Reader Service Card

PJ CODE

Code disks from six issues, only $39.95*

PROGRAMMER'S JOURNAL
CODE DISKSCRIPTION—

Subscribe Today!
You'll receive your code disks in the mail every other month throughout the year. Learn from *PJ* code or use it your own programs. Each disk contains all of the program listings published in *PJ* plus .EXE files, .OBJ files, benchmark source files, all for **only $39.95!***
If purchased separately, the regular price for these six disks is $60.00.
So save yourself some time, effort and over $20.00—order your *PJ DiskScription* today!

For faster service with VISA/MC call toll-free:

800-234-0386

* Prepaid, US only. Call or write for prices outside the US.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Celebrating our 10th Anniversary!

AMX™ 68000

KADAK's engineers bring years of practical real-time experience to over 600 installations world-wide.

This *real time* MULTITASKING KERNEL simplifies *real life* product development

- No royalties
- Full source code included
- * C interface
- Preemptive scheduler
- Intertask messages
- Terminal Handler
- * Dynamic memory allocation
- * Event Manager
- * Semaphore Manager
- * Resource Manager
- * Buffer Manager

AMX 68000 operates on any 68000/10/20 system

Manual only **$75 US**
AMX 68000 **$1600 US**
* Options — ask for price list.
(Shipping/handling extra)

Also available for 8080, Z80, 8086/186/286.

KADAK Products Ltd.
206 - 1847 W. Broadway
Vancouver, B.C. Canada V6J 1Y5
Telex: 04-55670
Fax: (604) 734-8114
Telephone: (604) 734-2796

Number **211** on the Reader Service Card

EVERLOCK

EVERTRAK

COPY PROTECTION

Designed for clone compatibility & strength

- Thwarts ALL Bit-copy Software
- Protect any COM/EXE w/o Source changes
- Shut down Debug Tracing & Disassemblers
- Supports all Hard & floppy Disk formats
- Restrict or allow Network Installs
- Limit program use by Date or #Executes
- Remotely reset Installs, Date or #Executes
- No damaged media or I/O plugs

SECURITY

If you HATE Copy Protection, but still want to protect your program from hackers and *reverse engineering*, **EVERTRAK** allows you to:

- Protect any COM/EXE w/o Source changes
- Shut down DEBUG systems and disassemblers
- Embed/display hacker-proof Customer information
- Restrict use by date-range and/or media-type
- Track unauthorized copies back to their source
- Totally invisible to End Users

For IBM & Compatibles. 30 day money back guarantee.
Free information and demonstration disk available.

TeleCheck VISA MasterCard

Az-Tech Software, Inc.
305 East Franklin
Richmond, MO 64085

(800) 227-0644 • (816) 776-2700 • FAX (816) 776-8398

Number **51** on the Reader Service Card

TOTAL CONTROL with *LMI FORTH*™

For Programming Professionals:

an expanding family of compatible, high-performance, Forth-83 Standard compilers for microcomputers

For Development: Interactive Forth-83 Interpreter/Compilers

- 16-bit and 32-bit implementations
- Full screen editor and assembler
- Uses standard operating system files
- 400 page manual written in plain English
- Options include software floating point, arithmetic coprocessor support, symbolic debugger, native code compilers, and graphics support

For Applications: Forth-83 Metacompiler

- Unique table-driven multi-pass Forth compiler
- Compiles compact ROMable or disk-based applications
- Excellent error handling
- Produces headerless code, compiles from intermediate states, and performs conditional compilation
- Cross-compiles to 8080, Z-80, 8086, 68000, 6502, 8051, 8096, 1802, and 6303
- No license fee or royalty for compiled applications

For Speed: CForth Application Compiler

- Translates "high-level" Forth into in-line, optimized machine code
- Can generate ROMable code

Support Services for registered users:

- Technical Assistance Hotline
- Periodic newsletters and low-cost updates
- Bulletin Board System

Call or write for detailed product information and prices. Consulting and Educational Services available by special arrangement.

Laboratory Microsystems Incorporated
Post Office Box 10430, Marina del Rey, CA 90295
Phone credit card orders to: ***(213) 306-7412***

Overseas Distributors.
Germany: Forth-Systeme Angelika Flesch, Titisee-Neustadt, 7651-1665
UK: System Science Ltd., London, 01-248 0962
France: Micro-Sigma S.A.R.L., Paris, (1) 42.65.95.16
Japan: Southern Pacific Ltd., Yokohama, 045-314-9514
Australia: Wave-onic Associates, Wilson, W.A., (09) 451-2946

Number **63** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Embedded SQL for a Relational Database

by
Richard Riehle

One of the more fascinating aspects of software is the almost continual introduction of new ideas. Some ideas are endowed with lasting virtue, while others become popular because of the prestige of their origin. For example, both relational database (RDB) technology and Structured Query Language (SQL) originated at IBM; no one can fault either for its pedigree. But while RDB has been praised widely throughout the computing industry as one of the bright new ideas for the flexible management of large databases, SQL has been somewhat controversial.

There are other methods besides SQL for accessing and maintaining RDB data structures. For simple conversational interaction with relational databases, I like a technique called Query By Example (QBE). QBE was originally supported by IBM but is being supplanted by SQL. One excellent implementation of QBE is in a product called Paradox, from Borland. A variation of QBE is also available under R:BASE from Microrim.

Relational database technology was popularized with the introduction of the dBASE series of products from Ashton-Tate, first on CPM machines and then under MS-DOS. Philistines debated the relational purity of dBASE, but it was a positive step toward relational thinking.

One of the difficulties with RDB products has been the awkward imperative languages supplied by the software manufacturers and the hurdles associated with interfacing the code to existing algorithms in other languages. Although remarkable products have been written in dBASE III, these products are often limited in scope and must be regarded as sophisticated prototypes of the future "real" system. This is not meant to denigrate either dBASE or Paradox—I have great respect for both and for the software practitioners who build elaborate systems using them.

For many software developers, though, the power of popular database products is not sufficient to meet the real needs of a particular client or marketplace. These developers have had two basic choices:

- Create a "roll-your-own" database management system (DBMS).
- Use a commercially available embedded DBMS product.

An example of the latter that I like is called Btrieve. [*See "Btrieve for C Programmers" in this issue.—Ed.*] For many professional software developers there has been no choice but to "roll their own." However, many developers would prefer a "loosely coupled" database-management product as part of the language environment they use. With the imperative languages of dBASE III, Paradox (PAL), or R:BASE, one must learn a completely new language, with all its peculiarities. Moreover, in dBASE III the procedural commands range from counterintuitive to nonsensical. The most recent version of R:BASE has SQL syntax, and the folks at Microrim seem to have a real interest in providing good SQL capability. Ashton-Tate's SQL feature for dBASE IV is reportedly fraught with bugs, but I must confess I have not used the SQL features of dBASE IV.

The ideal would be the ability to code a product in a standard, well-designed programming language but take advantage of standard methods of managing databases. For example, C is not the greatest database-management language ever invented, but it does possess enormous algorithmic power. The ability to combine that algorithmic power with a powerful database-management facility results in a very effective software system.

This article will deal with both SQL and relational databases. I chose to focus on SQL because it is gaining such wide acceptance throughout the world of RDB technology. There are implementations of SQL on mainframes, minicomputers, and microcomputers. The SQL syntax in all of these environments is essentially the same. In other words, SQL is a de facto standard, although there are some minor variations between implementations.

SQL Modes

There are two modes of operation for SQL: *interactive* and *embedded*. Interactive SQL (ISQL) is typically used by nontechnical people, such as accountants, staff assistants, and researchers, who have little need for elaborate reports and data processing. Embedded SQL is of interest to programmers because it can greatly enhance the software development process.

Embedding allows us to put SQL statements directly into the source code of our favorite programming language, which is called the *host*. This is of great benefit in manag-

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ing large, complex databases, since it provides an easy marriage to other software development tools. We can take simple SQL statements and envelop them within some other programming language source code. The implications of this are substantial. Whether you code in C, Pascal, Ada, COBOL, or any other language, SQL becomes an integral part of your code and simplifies the data-management procedures within that code.

Since this is a journal for programmers, the rest of this article focuses on embedded SQL (ESQL) rather than conversational ISQL. As software developers, you can take advantage of ESQL to build the code for your projects more rapidly and reliably. In addition, because ESQL raises the level of abstraction in your code, you should find it easier to adhere to good principles of software engineering.

Although ESQL can be embedded in many different host languages, most of my examples will be in COBOL; a few will be in Pascal and C. If you code in Ada, Modula-2, PL/1, or FORTRAN, you will find the same ESQL principles apply. However, the other languages will have modest and easy-to-understand differences, such as the variation in scope terminators for the EXEC SQL statement.

This article is not intended to be a complete tutorial on SQL or RDB. I can give you only a taste of both—and only in vanilla, chocolate, and strawberry. For a full portion of the more exotic flavors, try one of the books on RDB by C. J. Date [*See "Miscellaneous" —Ed.*] or James Martin. I shall explain key points of SQL and RDB as necessary in pursuit of an understanding of the main features of ESQL.

The Relational Database Concept

The concept of relational databases was introduced around 1970 by Dr. E. F. Codd, a mathematician at IBM. Since then it has been implemented in a variety of forms. Codd's idea of a relational database structure contains some important rules for determining whether an implementation is truly relational:

- The database is organized in a two-dimensional, rectangular array of rows and columns. A row is called a *tuple* and a column is called a *domain* or, sometimes, an *attribute*. The rectangular array is called a *table* or, more precisely, a *relation*.
- A row (tuple) contains no repeating groups.
- The entries in a column (domain) represent items of the same type. The word *type* can be thought of in the same way here as in programming.
- Each column (domain) has a unique name.
- Duplicate rows (tuples) are prohibited. Each row is distinct from all others.
- A row (tuple) of five entries is called a 5-tuple, one of 12 entries is a 12-tuple, and one of *n* entries is an *n*-tuple.

Every tuple in a relation has a unique entry that serves as its key. This is called its *primary key*. A tuple can have more than one unique entry, but it can have only one primary key. The other unique entries are referred to as *candidate keys*—that is, they are candidates to become a primary key. The primary key must be unique and nonredundant. Sometimes it is necessary to include more than one column (domain) in a primary key to make it unique. (From here on, I shall use the terms *table* and *relation, tuple* and *record,* and *column* and *domain* interchangeably.)

In RDB we often hear the term *normal form*. This refers to the "normalization" of the database and is a technique for creating new relations from repeating groups (such as those found in a COBOL OCCURS clause). A well-designed relation will not contain array entries. Each array would become a relation of its own with a key from the parent relation. This last characteristic can produce a slight redundancy of data between relations but preserves the integrity and nonredundancy of each relation. Moreover, it means that a relation can be modified (by the addition of columns, for example) without influencing the consistency or integrity of the database. Best of all, such modifications can be transparent to the logic of the applications programs.

If you have been working with database management systems, you may notice that not all relational database systems are truly relational when measured against Dr. Codd's criteria. However, it is beyond the scope of this article to critique the various products in this regard.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Relation Name = CUSTOMER

ACCOUNT	NAME	CODE	LOCATION
M1	CHIN	30	BEIJING
M2	NGUYEN	32	DA NANG
M3	SHAHANI	40	CALCUTTA
M4	ADACHI	30	NARA
M5	PAWANUI	40	KAAAWA
M6	FANENI	55	SAMOA
M7	NOORS	30	KABUL
M8	EMAYO	44	VIGAN

Relation Name = PRODUCT

ID	NAME	COLOR	FABRIC	LOCATION	PRICE
B1	HAT	MAUVE	FELT	SAMOA	40.00
B2	BLANKET	BLUE	WOOL	KABUL	60.00
B3	SHIRT	ORANGE	COTTON	CALCUTTA	10.00
B4	SHOES	BROWN	ALLIGATOR	SAMOA	14.00
B5	SHOES	WHITE	LEATHER	KABUL	18.00
B6	SUIT	WHITE	LINEN	SAMOA	45.00
B7	SCARF	GREEN	SILK	BEIJING	12.00
B8	SKI JACKET	BLUE	NYLON	VIGAN	65.00
B9	SWEATER	GRAY	WOOL	NARA	25.00
B10	PANTS	BLACK	COTTON	DA NANG	16.00
B11	HAT	TAN	PALM	KAAAWA	8.00

Relation Name = ORDERS

ACCOUNT	ID	QTY_ON_ORDER
M1	B1	20
M	B2	30
M1	B3	0
M1	B4	60
M1	B5	50
M1	B6	90
M2	B1	80
M2	B2	70
M2	B5	40
M3	B2	0
M3	B2	20
M4	B2	40
M4	B4	0
M4	B9	10

Dr. Codd went beyond simply defining a database made of tables. He also provided a solid theoretical foundation for RDB based on the mathematics of sets and groups and the range of operations for a relational database. The range of operations may be thought of as a "relational algebra." The basic operations permitted in this algebra consist of union, intersection, difference, Cartesian product, select, join, divide, and project. The first four operations are analogous to their counterparts of the same names in set theory but behave just a little differently.

Because this article is directed toward practical applications of embedded SQL, I shall avoid extensive digressions into the theory and mathematics of RDB. Substantial literature is available for those of you who wish to explore a more rarefied discussion of the topic. You may even want to read Dr. Codd's original paper, "A Relational Model of Data for Large Shared Data Banks," in *The Communications of the ACM* (either June 1970 or January 1983).

To make the rest of the material in this article somewhat more practical, I define three relations and discuss them in terms of SQL and relational operations. The relations are given in the three tables above. We will use these relations to look at some examples of embedded SQL code.

To complete the data definitions required for our examples of embedded SQL, I define the COBOL WORKING-STORAGE DIVISION entry as shown in Listing 1. As you can see, the third relation (ORDERS) is directly related to the information in the first two relations (CUSTOMER and PRODUCT). Also notice the names duplicated in the first two tables. In our SQL statements we use *dot qualification* to refer to items with duplicate names. For example, CUSTOMER.NAME refers to NAME in the CUSTOMER table, and PRODUCT.NAME refers to the name in the PRODUCT table. The practice of dot qualification is now widely used throughout the software industry, and many of you have probably seen it before. It is of special value in embedded SQL, since it improves program understandability and unique accessing of data elements.

Notice that our relations consist of tuples and domains. Dr. Codd tells us that for any tuple there is an element in some domain; that is, each column intersects with a row. At each intersection is a component of the table referred to in RDB as an *atomic value*. By "atomic," we mean that it is impossible—within the rules of SQL—to further decompose this component. Our tables seem to be in a specific order, but this is not a requirement of RDB.

SQL is different from the von Neumann (procedural) languages that we have been using for the last thirty or so years. We call SQL a "nonprocedural" language. Rather than operating on a one-for-one basis like FORTRAN, BASIC, C, or Ada, SQL operates on sets of data. In my article on Ada in Volume 6.6 of *PJ*, I discussed "abstract data types" and the idea of raising the level of abstraction. SQL raises the level of abstraction by allowing us to specify the "what" of our problem without regard to the "how." In SQL we deal with objects rather than algorithms.

Listing 2 gives a COBOL example using the tables described above. The example includes a statement in the COBOL program that accessed our relation, conditionally selected atomic values (WHERE PRODUCT.FABRIC = "MAUVE"), and stored the result in our WORKING-STORAGE entries declared earlier.

The use of data names prefixed with a colon (:) tells SQL to find a data declaration of that name in our program and store the result of the SELECT in the corresponding main memory address. An item with a colon prefix is called a *host program variable*, where the "host program" means the source program (COBOL in this example) that includes the SQL statements. A host program variable must be defined somewhere in the host program and must be spelled exactly as it is in the SQL statement.

In the example, the SELECT...WHERE found only a single matching table entry. Because SQL operates on sets of data, we need a way to deal with the possibility that we may have more than one hit on our conditional entry.

Any SQL statement that can be resolved interactively (ISQL) can also be used in ESQL. The reverse, however, is not true; some SQL statements are reserved for use in ESQL. These include:

```
EXEC SQL BEGIN DECLARE SECTION  END-EXEC.
EXEC SQL END   DECLARE SECTION  END-EXEC.
EXEC SQL       DECLARE cursor-name FOR
    SELECT ...   END-EXEC.
EXEC SQL       OPEN cursor-name  END-EXEC
EXEC SQL       CLOSE cursor-name END-EXEC.
EXEC SQL       FETCH cursor-name
    INTO ...   END-EXEC.
```

We shall use these statements as we examine ESQL.

The programmer developing an application using embedded SQL needs to be just as careful about design methods as he or she would be in any other language. In this regard, there are exception-handling methods and concurrent database-access capabilities in ESQL that a good programmer will incorporate into every program.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ESQL provides the WHENEVER exception condition:

```
EXEC SQL WHENEVER condition-name some-action
END-EXEC.
```

where

```
condition-name is  { NOT FOUND   | SQLWARNING
                                 | SQLERROR}
```

and

```
some-action is  { CONTINUE
                | GO TO     :host program variable
                | PERFORM   :host program variable
                | CALL      :host program variable}
```

where CONTINUE = no program action, proceed to next statement; GO TO and PERFORM = standard COBOL statements; CALL = subprogram CALL with parameters; and :host program variable = a COBOL paragraph or SECTION. Listing 3 is a typical example of the WHENEVER coded in COBOL. Similar facilities are available in other host languages. If we want to temporarily disable the WHENEVER function, SQL allows us to code it with a null action. Therefore,

```
EXEC SQL WHENEVER SQLERROR END-EXEC
```

will disable error checking.

Another important error-checking feature that should be included in your ESQL program is the INCLUDE SQLCA, the SQL communications area. The syntax is

```
EXEC SQL  INCLUDED SQLCA  END-EXEC
```

and it is coded in the WORKING-STORAGE SECTION of a COBOL program. Some implementations do not require the programmer to explicitly code this statement. Of particular importance is a field in SQLCA name SQLCODE, which is updated after the execution of every SQL statement. The values in this field may vary from one implementation to another, but in general an SQLCODE value of 0 means there were no errors. The SQLCODE is an analog of the FILE-STATUS code in the ENVIRONMENT DIVISION of an ordinary COBOL program.

In our earlier example of retrieving a record with the color mauve, we raised the possibility of our SELECT statement encountering more than one record in our relational table. The problem of duplicate records is a direct outcome of referencing sets of data rather than using the one-by-one record access so common to the usual programming. SQL provides a mechanism to solve this problem.

For our next example, we continue to use the sample tables and WORKING-STORAGE entries defined above. The results of using COBOL-85 for our example are shown in Listing 4, which illustrates a technique for accessing SQL using a COBOL-85 in-line PERFORM.

The standard variable name separator for SQL is an underscore (_), and in COBOL we use a hyphen (-). This is one of the real nuisances in using SQL with COBOL: I

MISOSYS

Programmer's Journal downloads are now available from MISOSYS' forum on Compuserve: GO PCS49

HartFORTH-86™

HartFORTH is a full 79-STANDARD FORTH. It is a Direct Threaded Code implementation which provides an execution speed between 10% and 40% faster than the classical Indirect Threaded Code implementation. HartFORTH is designed to run under DOS so the Virtual Memory that it accesses for storage and retrieval purposes is a file created and controlled by the operating system. Enhancements to the 79-STANDARD have been built into the HartFORTH kernel in the form of functions to call the DOS file handling routines so that other files may be created and accessed if required. A library of standard screens is supplied with HartFORTH to provide often used extensions to the language, such as double length and floating point math, editing of source screens, string manipulation, arrays, etc.

Here's the FINE PRINT

Any MISOSYS software product purchased directly from MISOSYS may be returned within 30 days of the invoice date for a full refund of the product purchase cost. Returns must be in a "like-new" saleable condition and be accompanied by a copy of the original invoice.

HartFORTH-86
$59.95 + $5S&H (US)

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS or 703-450-4181

- HartFORTH programs can invoke other programs via **EXEC** and **EXEC.PROG**.
- Functions create new files from within HartFORTH, and allow the current Virtual Memory file to be changed for another and manipulated at the individual block level.
- Provides the recommended 79-STANDARD DOUBLE NUMBER STANDARD EXTENSION word set that implements 32-bit operations.
- **CASE:** and **SWITCH:** functions allow multiway branching decisions to be taken with execution continuing in-line once the word branched to completes.
- String manipulators include: "VARIABLE, "CONSTANT, "!, "LEFT, "RIGHT, "MID, "+, "COMPARE, and ">.
- DOS software and hardware interrupt vector access support via: GET.VECTOR, PUT.VECTOR, THIS.SEG, DI, and EI.
- V24 program input, output, and interrupt input support.
- *Overlay management words:* FORGET.OVLY, OVLYNAME, PUT.DATA, OVLY.ENTRY, SAVE.OVLY, CORRECT?, LOAD.OVLY, NEW.OVLY, RUN.OVLY, and LEAVE.OVLY.
- Screens provide trigonometric functions: SIN., COS., TAN., SIN, COS, and TAN.

Number 121 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

always get my underscores and hyphens mixed up and have to go back and correct them.

Listing 4 also introduces the *cursor*. The cursor is somewhat like the NEXT clause in a dynamically accessed random COBOL file. With each FETCH, the cursor is advanced to the next record in the set of selected records. This solves the problem of SQL dealing with sets rather than individual records. The use of the cursor is fundamental to creating embedded SQL code. Combining it with the FETCH command, the programmer can access an entire set of records and process the set one record (tuple) at a time.

You will notice that the PRODUCT_CURSOR is declared in WORKING-STORAGE SECTION in association with a SELECT clause. At this point PRODUCT_CURSOR and the corresponding SELECT are merely declarations to the program. The SELECT is executed only after PRODUCT-CURSOR is open in paragraph 300-CHECK-PRODUCT. When the SELECT is executed, a temporary result table that is accessible by the FETCH is created. The programmer must keep in mind that the FETCH will access only one tuple at a time. FETCH cannot be advanced by 2 or 3 or some other number. It continues to behave like the READ...NEXT in DYNAMIC ACCESS METHOD of COBOL. Therefore, it is the responsibility of the programmer to control any special processing that must be performed on, say, every other tuple in the table.

In Listing 4 we declared PRODUCT_CURSOR using the

```
ORDER BY  domain
```

clause. The outcome of the SELECT will be a temporary table sorted in the order of the elements of the specified domain. The order may be specified as either ascending (ASC) or descending (DESC), just as in any ordinary SELECT statement. The programmer must be careful when using the ORDER BY clause, because it is incompatible with certain other cursor-controlled operations, such as those that follow.

The cursor provides other capabilities by letting us reference its current location. In our program we could have had a statement such as

```
EXEC SQL UPDATE PRODUCT
     SET PRODUCT.COLOR = "BEIGE"
     WHERE CURRENT OF CURSOR PRODUCT_CURSOR;
END-EXEC.
```

or, if we had declared a cursor named ORDER-CURSOR for our ORDERS table and had a DATA DIVISION entry named NEW-SALE,

```
EXEC SQL UPDATE ORDERS
     SET ORDERS.QTY_ON_ORDER = QTY_ON_ORDER +
          :NEW-SALE
     WHERE CURRENT OF CURSOR ORDER_CURSOR
END-EXEC.
```

We can also use the current cursor location to delete records, provided we have not used the ORDER BY clause:

```
EXEC SQL DELETE FROM CUSTOMER
         WHERE CURRENT OF CUSTOMER_CURSOR
END-EXEC.
```


"Quick and Clean®" COBOL

COBRA® – THE COBOL REPORT ACTIVATOR SYSTEM

What is the COBRA System?

The COBOL REPORT ACTIVATOR (*COBRA*) System is a simple, quick and powerful method used to generate error free structured COBOL programs. It is an ideal tool for ad-hoc reports, file reformatting and multiple program systems. *COBRA* may be used with most COBOL compilers.

Because the *COBRA* generated COBOL code is structured and well written, it is easy to follow and change. Multiple program systems are easily maintained. Reports created by the *COBRA* system may vary based on user demand, but will contain standardized features. Control break and total reports are easily produced, files may be automatically sorted, and records may be selected only when specific conditions exist.

Each time a *COBRA* system report is created, an error-audit report is also created. It shows statistics on records read, selected, rejected because of invalid data, number of detail lines printed, and when necessary, the number of output records for each file. The first 80 characters of invalid records are printed, along with a message giving the reason for rejection.

How does the COBRA System work?

The *COBRA* system is a 'keyword driven' system - that is, each statement begins with a keyword. It is also freeform; any statement may start in any column, and statements may be in any order. The COBOL code is generated based on your selection of keywords.

The COBRA System:

* Is easy to learn and use
* Produces efficient structured COBOL code
* Uses saved file descriptions
* Accepts your COBOL code and COPY members
* Can create five output files per program
* Provides for record selection
* Sorts in ascending, descending, or mixed sequences, or, has a no-sort option
* Supports Indexed and Sequential files
* Provides extra report options, including:
  - printing the TIME the report was produced
  - putting dashes above and/or below column headings and before the final report line
* Allows site defined defaults for efficiency and standards, including:
  -Input record length up to 9995 characters
  -Report page width up to 160 characters
  -Report spacing flexibility (single, double, or triple spaced)
  -Report date format
  -Whether to print *COBRA* source statements in the generated COBOL code
* Offers a money-back guarantee

List Price $495 • For more information please contact:

COBRA Systems Software

505 Acadia Drive P.O. Box 750817
Petaluma, CA 94975 • (707) 763-5123

Number 281 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

So far, this article has concentrated on SQL as it is used in COBOL and COBOL-85. Listings 5 and 6 provide examples of SQL as it might be embedded in other languages.

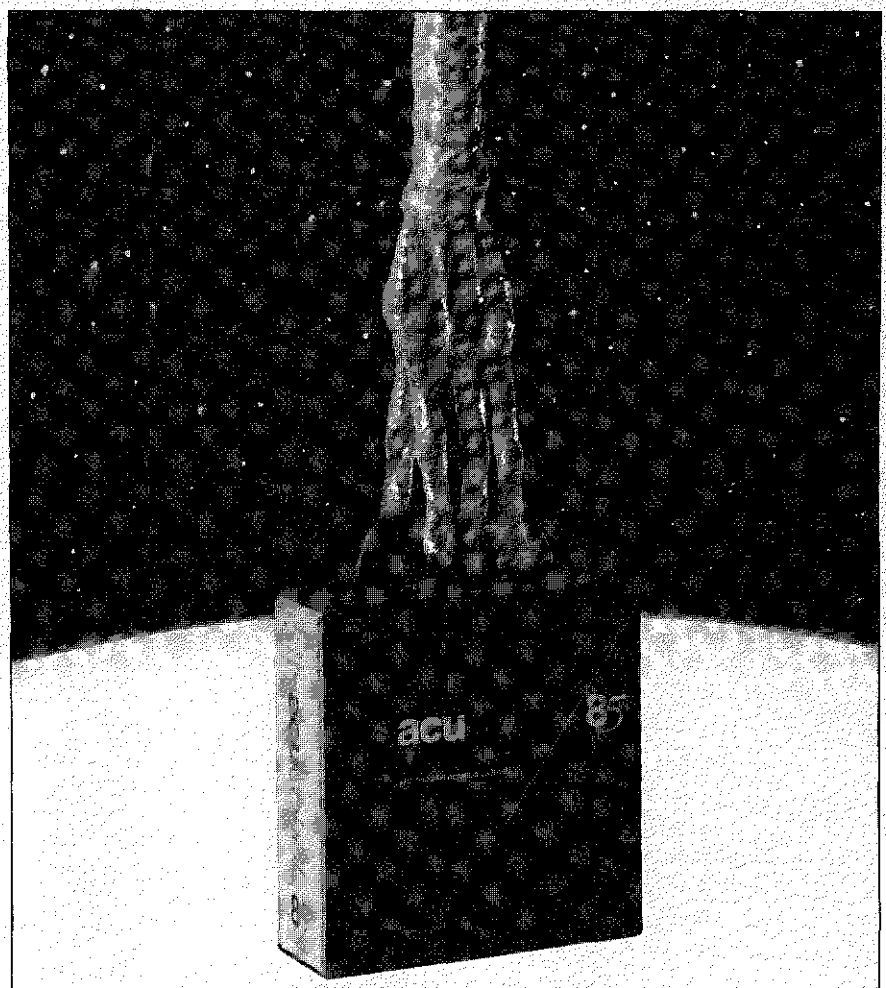
As with the COBOL examples shown earlier, the Pascal program data names are referenced in the SQL portion of the program using the colon (:) prefix. Notice, too, that we do not have the clause END-EXEC; this clause is used only in COBOL. In Pascal, PL/1, and C, the semicolon (;) serves as the equivalent of the END-EXEC clause. In Pascal and C, the SQL INSERT command allows us to insert a new tuple into the relation.

There are implementations of SQL that can be used with virtually every popular programming language. As of this writing, IBM and INGRES seem to have the most complete implementation of embedded SQL, but ORACLE is not far behind. Nearly everyone who supports embedded SQL supports it under COBOL. Others support Pascal, C, PL/1, and FORTRAN. INGRES supports Ada. So far as I know, no one supports a FORTH implementation. IBM supports embedded SQL under all of the languages mentioned and adds REXX, BASIC, APL, Prolog, and mainframe assembly language. Novell, the owners of Btrieve and other database-management products, has been advertising its XQL recently. This sounds like a good implementation of embedded SQL, but I have not had an opportunity to use it. Someone out there might have some good words to say about XQL, and I'd like to hear from you.

If you are developing database-intensive applications and need a quick, effective, and maintainable approach to your problem, ESQL could be just the ticket, especially if you also need to develop complex algorithms that are beyond the reasonable limits of dBASE, R:BASE, or Paradox. PJ

Richard Riehle is an instructor in Ada at Merritt College in Oakland and Foothill College in Los Altos, California. He is also an independent computer consultant. Mr. Riehle's training organization, ICE (Intensive Computer Education), conducts seminars on a variety of computer subjects. Mr. Riehle has over twenty years' experience in software development. He is currently working on a book, From COBOL to Ada, *a tutorial for experienced business-programming professionals. Readers may write Richard at 1090-105 Tanland Drive, Palo Alto, CA 94303, or phone him at 415-858-1267.*

Code follows.



INTRODUCING
THE MOST PORTABLE COBOL
IN THE UNIVERSE

Go where you want to go
Transport COBOL programs to almost any platform—without changing one line of code. Run on AIX, MS-DOS, NETBIOS, UNIX, OS/2, ULTRIX, XENIX and VMS without recompiling. No other COBOL-85 offers such hardware independence, or the instantly-expanded possibilities.

Open new windows in your applications
Create better-looking programs with windows. Add help screens, pop-up/pull-down menus, context-sensitive messages, more. ACUCOBOL-85 has a Display Window verb, including extensions for boxed, reverse-video and pop-up windows in color. Paint special forms on the screen with our Display Line verb. And scroll up/down inside windows.

Don't get lost in space:
ACUCOBOL-85 keeps you on course
ACUCOBOL-85 is NBS certifiable error-free under the High 1974 and Intermediate 1985 ANSI standards. If you lose course, it has its own interactive source code debugger that's fast and easy to use.

Step into the future
Make your job simpler and stay competitive—even ahead of the game. Put the next generation COBOL to work for you today.

Free evaluation copy
Call (619) 271-7097 today for your free evaluation copy of ACUCOBOL-85. And see the shape of things to come.

acucobol inc

Voice: (619) 271-7097 Fax: (619) 566-3071
7950 Silverton Avenue Suite 201 San Diego California 92126

ACUCOBOL-85/the ACUCOBOL logo are trademarks of ACUCOBOL, Incorporated; MS-DOS/XENIX of Microsoft Corporation; NETBIOS/OS/2/AIX of International Business Machines Corporation; VMS/VAX/ULTRIX of Digital Equipment Corporation; UNIX of AT&T Bell Laboratories; RM/COBOL and RM/COBOL-85 of Ryan-McFarland Corporation.

Number 282 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "Embedded SQL . . ." by Richard Riehle

1

```
LISTING 1
DATA DIVISION.
WORKING-STORAGE SECTION.
EXEC SQL  BEGIN DECLARE SECTION  END-EXEC.
01 ORDER-ENTRY-WORK-AREA.
   05 PRODUCT-ID          PIC X(03).
   05 PRODUCT-NAME        PIC X(20).
   05 PRODUCT-COLOR       PIC X(10).
   05 PRODUCT-FABRIC      PIC X(10).
   05 PRODUCT-PRICE       PIC 9(02)V99.
   05 LOCATION            PIC X(20).
   05 CUSTOMER-ID         PIC X(03).
   05 CUSTOMER-NAME       PIC X(20).
   05 CUSTOMER-LOCATION PIC X(20).
   05 TARIFF-CODE         PIC X(02).
EXEC SQL END DECLARE SECTION END-EXEC.

LISTING 2
PROCEDURE DIVISION.
000-BEGIN.
          PERFORM 001-ACCESS-DATA-BASE.
        . . .

001-ACCESS-DATA-BASE.
          EXEC SQL
            SELECT PRODUCT.ID,
                   PRODUCT.COLOR,
                   PRODUCT.LOCATION
               INTO
                   :PRODUCT-ID,
                   :PRODUCT-COLOR,
                   :PRODUCT-LOCATION
               FROM PRODUCT
               WHERE PRODUCT.FABRIC = "MAUVE"
          END-EXEC.
001-END-DB-ACCESS.
        EXIT.

LISTING 3
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 EXEC SQL  BEGIN DECLARE SECTION  END-EXEC.
 01 ORDER-ENTRY-WORK-AREA.
    05 PRODUCT-ID          PIC X(03).
    05 PRODUCT-NAME        PIC X(20).
    05 PRODUCT-COLOR       PIC X(10).
 EXEC SQL END DECLARE SECTION END-EXEC.
 SCREEN SECTION.
* (SCREEN SECTION is not standard COBOL but is included
* in many microcomputer COBOL compilers)
 01 MAIN-SCREEN.
    05 SC-PRODUCT-ID       PIC X(03).
    05 SC-PRODUCT-NAME     PIC X(20).
    05 SC-PRODUCT-COLOR    PIC X(10).
 PROCEDURE DIVISION.
 000-BEGIN.
     EXEC SQL WHENEVER SQLWARNING
          GO TO :900-WARNING-ROUTINE
     END-EXEC.
     EXEC SQL WHENEVER SQLERROR
          GO TO :920-ERROR-ROUTINE
     END-EXEC.
     PERFORM 100-MAIN-ROUTINE UNTIL some-condition.
     PERFORM 800-TERMINATION-ROUTINE.
     STOP RUN.
 100-MAIN-ROUTINE.
     PERFORM 200-GET-SCREEN-INPUT.
     MOVE SC-PRODUCT-ID TO PRODUCT-ID.
     MOVE SC-PRODUCT-NAME TO PRODUCT-NAME.
     MOVE SC-PRODUCT-COLOR TO PRODUCT-COLOR.
     EXEC SQL
        INSERT  INTO PRODUCT (ID,
                              NAME,
                              COLOR)
                      VALUES  (:PRODUCT-ID,
                               :PRODUCT-NAME,
                               :PRODUCT-COLOR)
     END-EXEC.
************  additional main routine code **************
 200-GET-SCREEN-INPUT.
 800-TERMINATION-ROUTINE.
 900-WARNING-ROUTINE.
 920-ERROR-ROUTINE.

LISTING 4
WORKING-STORAGE SECTION.

     . . .

     EXEC SQL  DECLARE  PRODUCT_CURSOR CURSOR FOR
               SELECT
                     PRODUCT.ID,
                     PRODUCT.NAME,
                     PRODUCT.COLOR,
                     PRODUCT.FABRIC
               FROM  PRODUCT
               WHERE PRODUCT.ID >= :PRODUCT-ID
               ORDER BY PRODUCT.ID
     END-EXEC.
PROCEDURE DIVISION.
000-MAIN.
    PERFORM 001-INITIALIZE.
    PERFORM 300-CHECK-ON-PRODUCT.
    PERFORM 900-END-PROGRAM.
300-CHECK-PRODUCT.
     EXEC SQL OPEN PRODUCT_CURSOR END-EXEC.
     PERFORM UNTIL SQLCODE = +100
             EXEC SQL
                  FETCH PRODUCT_CURSOR
                    INTO  :PRODUCT-ID,
                          :PRODUCT-NAME,
                          :PRODUCT-COLOR,
                          :PRODUCT-FABRIC
             END-SQL
             PERFORM 600-VALIDATE-ROUTINE
             IF VALIDATE-RETURN-CODE = 0 THEN
                ADD 1 TO VALID-RECORD-COUNT
                PERFORM 400-PRODUCT-UPDATE-ROUTINE
             ELSE
                ADD 1 TO NON-VALID-RECORD-COUNT
             END-IF
     END-PERFORM
     EXEC SQL CLOSE PRODUCT_CURSOR END-EXEC.

400-PRODUCT-UPDATE-ROUTINE.
          . . .

600-VALIDATE-ROUTINE.
          . . .

900-END-PROGRAM.
```

2

Code continues . . .

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Virtual Memory for DOS!

MicroWay is your best source for the software and hardware you need to get true 32 bit performance from your 386. These include 32-bit tools, such as NDP Fortran and C, and the 32-bit applications that were developed with them (see last paragraph). These products run in protected mode under Unix, Xenix, or Phar Lap extended MS-DOS.

Starting with release 1.4VM, NDP Fortran and C not only access 4 gigabytes of memory, but run with Phar Lap's new VMM extension which provides 386 protected mode virtual memory. Now you can run a program with a 30 MB array on a 2 MB system simply by having 30 MB of free space on your hard disk.

MicroWay also offers transputer based parallel processing boards and languages that run in an XT, AT, or 386. Each of the T800 RISC processors on these boards packs the power of a 20 MHz 386/1167. Our best selling board, the Quadputer2™, has four T800s and boasts 40 MIPS/6 megaflops of processor throughput.

Dr. Robert Atwell, leading defense scientist, calculates that NDP Fortran-386 is saving him $12,000 per month in rentals of VAX hardware and software while doubling his productivity!

Fred Ziegler of AspenTech in Cambridge, Mass. reports, "I ported 900,000 lines of Fortran source in two weeks without a single problem!" AspenTech's Chemical Modeling System is in use on mainframes worldwide and is probably the largest application to ever run on an Intel processor.

Dr. Jerry Ginsberg of Georgia Tech reports, "My problems run a factor of six faster using NDP Fortran-386 on an mW1167 equipped 386/20 than they do on my MicroVAX II."

MicroWay manufactures Weitek 1167 and 3167 coprocessor cards that run with the 80386. Both cards include an 80387 socket. The 1167 is 2 to 4 times faster than the 80387. The 3167 runs 30% faster than the 1167 in double precision. The key to achieving this speed increase is our NDP Fortran or C and the new 32-bit applications that offer Weitek support. Either processor provides a dramatic increase in throughput for graphics intensive applications. These include VersaCad and Hoops 3D graphics, ANVIL 5000 CAD/CAM, SRAC and Swanson Analysis finite element packages, Mathematica and a host of other packages that were recently ported to the 386 using our NDP Fortran and NDP C. Please call (508) 746-7341 for more information.

32-Bit Compilers and Tools

NDP Fortran-386™ and **NDP C-386™** Compilers generate globally optimized, mainframe quality code. Both run in 386 protected mode under Unix, Xenix or Phar Lap extended MS-DOS. The memory model employed uses 2 segments, each of which can be up to 4 gigabytes. They generate code for the 80287, 80387, mW3167 or mW1167. Both include high speed EGA graphics extensions written in C that perform BASIC-like screen operations.

- **NDP Fortran-386™** Full implementation of FORTRAN-77 with Berkeley 4.2 and Fortran-66 extensions. $595
- **NDP Fortran-386/VM** $695
- **NDP Fortran-386/UNIX/XENIX** . . . $795
- **NDP C-386™** Full implementation of AT&T's PCC with MS and ANSI extensions. . . . $595
- **NDP C-386/VM** $695
- **NDP C-386/UNIX** $795

Phar Lap 386/VMM extensions are supported by the VM releases of NDP Fortran and C. Virtual memory is used automatically by the compilers and the code they generate. This makes it possible to compile and run programs as large as the free space on your hard disk.
Phar Lap Virtual Memory Manager . . $295
Phar Lap Development Tools $495

NDP Windows™ — NDP Windows includes 80 functions that let you create, store, and recall menus and windows. It works with NDP C-386 and drives all the popular graphics adapters . . . . . Library: $125, C Source: $250

NDP Plot™ — Calcomp compatible plot package that is callable from NDP Fortran. It includes drivers for popular plotters and printers. Works with CGA, MDA, EGA and VGA . . . $325

NDP/FFT™ — Includes 40 fast running, hand coded algorithms for single and double dimensioned FFTs which take advantage of the 32-bit addressing of the 386 or your hard disk. Callable from NDP Fortran with mW1167 and 80387 support $250
387FFT for 16-bit compilers $250

NDP to HALO '88 Graphics Interface — This module enables you to call graphics routines in HALO '88 from NDP Fortran or C. $100

MicroWay® 80386 Support (508) 746-7341

Parallel Processing

Videoputer™
The highest performance graphics card on the market. Uses a T800 and TI 34010 in conjunction with a 130 MHz BrookTree DAC. The board comes with one meg of system RAM, one meg of video RAM and a library of graphics primitives. It runs standalone or in conjunction with a transputer network and drives 32 and 64 KHz analog monitors $4995

Monoputer2™
The world's most popular PC transputer development product now extends the memory available for developing transputer applications from 2 to 16 megabytes. The board features a DMA bus interface for fast I/O.
Monoputer2 with T414 (0 MB) $995
Monoputer2 with T800 (0 MB) $1495

Quadputer2™
This board for the AT or 386 can be purchased with 2, 3 or 4 transputers and 1 or 4 megabytes of memory per transputer. Two or more Quadputers can be linked together to build networks of up to 100 or more transputers providing mainframe power . . . . . **from $3495**

Transputer Compilers and Applications
These Parallel languages are designed for use with Monoputer2 and Quadputer2.
Logical Systems Parallel C $595
3L Parallel C or 3L Parallel Fortran . . $895
Microfield - Finite element analysis . . $1600
ParaSoft: Parallel Environment . . . $500
Performance Monitor . . . $500
C Source Level Debugger $500
T800/NAG™ (See NDP/NAG) $2750

387BASIC™ — Our 16-bit MS compatible compiler introduces numeric register variables to produce the fastest running 80x87 code on the market $249

Weitek-Based Coprocessor Boards

mW1167™ and **mW3167™** coprocessor boards are built at MicroWay using Weitek components. Each includes an 80387 socket.
mW1167-16 . . . . . $695
mW1167-20 . . . . . $895
mW1167 Microchannel-16/20 . . . . . CALL
3167-20 . . . . . $995
3167-25 . . . . . $1295
mW3167/80387 Board $150

Intel Coprocessors and RAM

8087 . . . . . $84
8087-2 . . . . . $120
80287-8 . . . . . $195
80287-10 . . . . . $220
80C287A (CMOS) $280
80387-16 . . . . . $350
80387-16SX $310
80387-20 . . . . . $400
80387-25 . . . . . $500
287Turbo-12 (for AT compatibles) . . . $350
RAMpak™-1 meg 32 bit memory module for Compaq system memory board . $495
256K 100ns DRAM $6.75
256K SIMMS . . . . . CALL
1 MB SIMMS . . . . . CALL
(All of our Intel coprocessors include 87Test.)

Multi-User Accelerators

MicroWay's **AT8™** and **AT16™** intelligent serial controllers run 8 to 16 terminals under Unix or Xenix without bogging down your AT, 80386 or PS/2 PC. **AT8**: $895 **AT16**: $1295

32-Bit Applications

PSTAT-386 — Popular mainframe statistics package. Full version was ported $1495

NDP/NAG™ — Features a library of 268 engineering and scientific numerical algorithms. Callable from NDP Fortran $895
Call for our list of new 32-bit applications.

12 MHz PC Accelerators

FastCACHE-286 12 MHz $299
SuperCACHE-286 12 MHz $399
FastCACHE-286 9 MHz $199

MicroWay

World Leader in PC Numerics

Corporate Headquarters: P.O. Box 79, Kingston, MA 02364 USA (508) 746-7341
32 High St., Kingston-Upon-Thames, U.K. 01-541-5466
USA FAX 508-746-4678 Australia 02-439-8400 Germany 069-75-1428

Number 104 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Riehle code, continued

LISTING 5
PASCAL SQL

```
{variable declarations}

exec sql
  begin declare section;
  var in_product_rec    : record;
      in_product_id     : fstring (3);
      in_product_price  : int32;
      in_product_name   : fstring (20);
  end;
exec sql
  end   declare section;

{additional Pascal code here}

. . .

begin
   in_product_rec.in_product_id    := 12;
   in_product_rec.in_product_price := 3000;
   in_product_rec.in_product_name  := "PANTS";
   exec sql insert into product
              (id , price, name )
            values
              (:in_product_rec.in_product_id,
               :in_product_rec.in_product_price,
               :in_product_rec.in_product_name);

end.
```

LISTING 6
The C Language

```
/* variable declarations  /*

exec sql
  begin declare section;
struct in_product_rec
    { char in_product_id;
      long in_product_price;
      char in_product_name[20];
    };
exec sql
  end   declare section; /*
/* additional C code here  /*

. . .

{
   in_product_rec.in_product_id    = 12;
   in_product_rec.in_product_price = 3000;
   strcpy (product_rec.in_product_name, "PANTS");
   exec sql insert into product
              (id, price, name )
            values
              (:in_product_rec.product_id,
               :in_product_rec.product_price,
               :in_product_rec.product_name);
}
```


C™ Programmer's Toolbox

PC/MS DOS 2.1 & later

CFLow™	Determine your C program's function hiearchy; external P/T functions; etc.
CPrint™	Beautify C source code to your selected format
PMon™	Monitor your program/OS execution; Determine where time is being spent...
Volume I	+ 12 more tools...

Are you creating better, faster, higher quality and easier to read programs in a fraction of the time?

Join our many beginning and expert users.

Volume II

9 more tools that you requested including:

CLint™	Perform additional syntax checks of your C source code-find silly mistakes...
CritPath™	Determine your C program's critical path from PMon and CFlow report inputs...
CXRef™	Trace your symbol/variable usage; identify potential usage problems...

The Toolbox works with all compilers and libraries. Specify any number of files through wildcards and/or indirect files.

Buy both Volumes I & II for $140, saving $20, or only one volume for $79.95. CA residents please add applicable tax. Overseas, please add $10 shipping & handling. With our 30 day money back guarantee, you can't lose.

Call or write us today

MasterCard VISA®

C™ Programmer's McTool Series

Apple® Macintosh™

Common Tool Features
- o Powerful, built-in editor
- o Automatic/manual search & replacement
- o Selective C source code highlighting
- o Works with all C compilers
- o ANSI support

#1 McCPrint™	$59.95
#2 McCLint™	$99.95

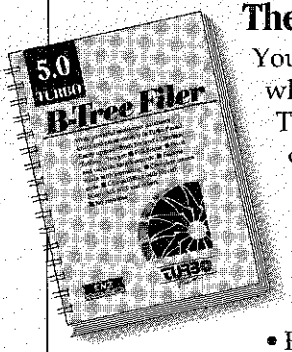
MMC AD Systems
Box 360845
Milpitas, California 95035
(408) 263-0781
"The C Tool Specialists"

Number 94 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

New! Turbo Pascal Tools for Powerful Multi-User Databases

Write powerful multi-user databases faster and easier using the language you love - Turbo Pascal - and **B-Tree Filer 5.0**. Easily upgrade your present applications that use Borland's Database Toolbox.



The Performance and Connections You Need

You'll have the fastest, safest, most flexible databases when you use **B-Tree Filer 5.0** – no rigid structure, no TSR hassles, no running out of files. And they're compatible with Novell, 3Com, MS-NET, and others.

The Complete B-tree Tool Kit

In over 100 Turbo 5.0 procedures you get:
• Fixed and variable length records • Two billion records per database • 100 keys per index file • Flexible record locking • B-tree indexing with advanced page balancing • Fail-safe mode with journaling • Units for sorting, browsing, reindexing, and reorganizing • Same source runs on any network • Complete source code. B-Tree Filer includes full documentation and technical support and you pay NO royalties.

B-Tree Filer 5.0 is only $125. (single user)
Multi-user version with network support, only $175.

Call toll-free for credit card orders.
1-800-538-8157 ext 830 (1-800-672-3470 ext 830 in CA)

Complete capabilities brochure available, call 408-438-8608 (9 to 5 PST) or write for more information. Also from TurboPower:
• Turbo Analyst 5.0, tune and document your programs, only $99.
• Turbo Professional 5.0, the best general purpose library, $125.

Satisfaction guaranteed or your money back within 30 days. Turbo Pascal 4.0 or 5.0 is required. Shipping and taxes prepaid in U.S. and Canada. Elsewhere add $15 per unit.

TURBO Power

TurboPower Software
P.O. Box 66747
Scotts Valley, CA 95066-0747

Number **25** on the Reader Service Card

Multitasking
for the IBM-PC/XT/AT
with
Multidos Plus

Ideal for developing process control, data acquisition, communications, and other applications which require a multitasking environment. Check these features which make Multidos Plus an unbeatable value:

- Run up to 32 programs concurrently
- Your software continues to run under PC-DOS or MS-DOS. No need to learn a new operating system.
- Use the compilers you already have. Supports software written in most compiled languages or assembly language.
- Operator commands to load and run programs, change priority, check program status, and to abort/suspend/resume task execution.
- Optional independent foreground and background displays.
- Programmatic interface through INT 15H for the following:
  * Intertask messaging to send, receive, check or wait on 63 different message queues.
  * Task control by means of semaphores. Get, release, check, or wait on semaphores.
  * Change task priority.
  * Suspend task for specified interval.
  * Spawn and terminate external and internal tasks.
  * Disable/enable multitasking.

Requires: IBM PC/XT/AT or true clone. MS-DOS (or PC-DOS) 2.0 or later. Enough memory to hold Multidos Plus (64KB) and all your application programs. You may also need 4 or 16KB of memory for "hidden screens" for each active task.

only: **$24.95** or

$99.95 with source code written in Lattice C and Microsoft Assembler

Call (800) 872-4566 ext. 350 for Visa and Mastercard ~ orders only. Technical information and all other orders call (508) 651-0091 or our 24-hour FAX at (508) 655-8860. Send check or money order (Drawn on US banks only) to:

Nanosoft
13 Westfield Road, Natick, MA 01760

Mass. shipments must add 5% sales tax.
Foreign orders add $5.00 shipping & handling.
COD orders add $5.00

Number **109** on the Reader Service Card

CHANGE OF ADDRESS?

Please send us your new address along with your previous ZIP code, using this handy form— Thanks.

NAME _____
COMPANY _____
ADDRESS _____
CITY _____ STATE _____
ZIP CODE _____ OLD ZIP CODE— _____
TELEPHONE _____

Please mail to:
Programmer's Journal
P. O. Box 30160
Eugene, OR 97403

Or call toll free
800-234-0386

PRODUCED 2024 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Btrieve for C Programmers

by

Amrit Khalsa and Pete Maclean

File management is one of the most important considerations in creating a software application. A record-management system, such as Btrieve, is designed to aid the programmer—freeing him or her from reinventing what Btrieve offers: a file-locking and record-locking system for stand-alone PCs or multi-user PC networks that are NETBIOS compatible.

Integrating Btrieve into a C application can be enormously time-consuming and boring. Btrieve provides a generic record-management tool, around which we sculpted a tool for interfacing with a C application system. It took forever and was not particularly fun. Since then we have found numerous programmers who have had to reinvent this wheel. Here now, is our interface, which should save you countless hours in interfacing Btrieve to your application.

Our approach to using Btrieve consists of two steps. The first is to simplify the structures that Btrieve requires while adding additional structures to keep track of information needed by the application program. Second, instead of having Btrieve called directly from various places within each program, we have constructed our own library of file-management functions that call the Btrieve function.

To ensure uniform code maintenance, our priority is to keep all of the file-handling code within a single module. Because Btrieve operations are invoked only from within this module, systemwide changes are easy to make; for example, implementing new Btrieve features, testing various record-locking techniques, or experimenting with variable-length versus fixed-length record formats are automatically used in all of the programs.

Data Management

The programmer must supply several data structures for each Btrieve-managed file. A basic file-definition structure defines the format of the file when it is created; Btrieve stores this in the data file's header record. A *position block* associated with each open file maintains the positions of the current, last, and next records, according to the current method of access. A set of key-definition structures defines each file access path, and so on.

Defining the Structures The Btrieve manual offers a section on programming in C, with instructions for using some of the library's functions. The manual could lead one to conclude that different structures are required for different operations. We discovered that we could define one key structure and one file structure that would be acceptable for all Btrieve operations that require them.

Keys, as defined in Btrieve, correspond to fields within the data record. For example, a key can be defined as starting at byte 0 for a length of 7 bytes. If a key is made up of more than one field, it is called a *segmented key.* For example, if a person's first and last names are kept as separate items in a record, then the two could be combined into a segmented key (Listing 1).

Other File Information In Listing 2, we define our own structure to hold the position block that Btrieve needs, along with the filename. Any other file-level information that is to be maintained for the application should be included in this structure. We also define a structure designed to maintain any information concerning the keys that fit into the key structure that Btrieve requires.

Organizing the Structures We set up the structures for our data files in parallel arrays as follows:

```
#define N_FILES 10

struct filespec btstruct[ N_FILES ];
struct btfhand  fh[ N_FILES];
struct btfkey   fkey[ N_FILES ];
```

This arrangement allows us to access information contained within any one of these structures by supplying an ordinal that can be used to index any array. If an extra file is added to the system, we need only change the value equated to N_FILES and enter information specific to that new file.

A series of **#define**s gives us tags for the various files in our system. For example:

```
#define   TABLEF     0    /* Table file */
#define   NAMEF      1    /* Name file */
#define   CODEF      2    /* Codes only file */
```

Designing the Application

With the design for the header file completed, we move on to construct our own library of file-management functions. Our basic set of file-handling functions was largely determined by the requirements of the Btrieve function call itself. Our second criteria for designing a limited set of functions was to segregate functions into file-level, record-level, and key-level processing. This approach has the advantage of isolating the file-locking and record-locking user-interface source within appropriate modules at expected locations. Each function tests for a limited subset of the possible error codes Btrieve returns.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Btrieve, a Novell product, is a record-manager system designed to aid PC programmers in the development of application packages. The heart of the product is a library of functions for managing record-structured files under DOS, OS/2, and XENIX. In addition to the fast access time of a B-tree indexing system, Btrieve offers record locking for multi-user and network systems that are NETBIOS compatible. Four programming languages—BASIC, Pascal, COBOL, and C—are supported.

Although conceptually the package comprises many functions, only a single function call is provided to the programmer. The first argument to this function identifies which of 33 possible operations is to be performed. Under DOS, most of Btrieve's work is done by a memory-resident module.

The syntax of the basic Btrieve function parameters is:

```
btstatus = BTRV( operation code, position block,
    data buffer, data buffer length, key buffer,
    key number );
```

The Btrieve function wants these arguments to be used in different ways for different operations. Sometimes, for instance, one must supply a filename in the data buffer, a drive name in the key buffer, or the address of the file structure in the data buffer. Sometimes, Btrieve requires only a position block and ignores everything else. [Refer to the "Btrieve Record Operations (Basic Programming)" section of the manual for explicit documentation as to the requirements for each operation code.]

We are not publishing the source code for certain Btrieve features that we feel are too specialized or too application dependent for this article. Specifically, we exclude the function for creating a supplemental index and the function for setting a file owner. We include no record-locking, file-locking, or error-handling source code. Our key buffers are exclusively strings. We do not discuss alternate collating sequence or the extended key types.

Checking Memory-Resident Status Btrieve is a memory-resident application. Listing 3 checks to see if it has been loaded—no quick and dirty here.

Initializing a File Each of our Btrieve-managed files has a unique initialization function called prior to a create or open operation. Listing 4 is an example of a file-initialization function for file number 0. Clearly, it would be more efficient to initialize the structures statically at declaration time. We chose to use a function, because it makes the initializations much more readable and hence easier to modify. Introducing a new file into the library is as fast as supplying an initialization function and adding 1 to N_FILES.

File Handling There are two file-handling functions: **bt_file** and **bt_openclose** (Listing 5). The **bt_file** function is called to create a file or ascertain file-status information. Creating a file does not open it; our **bt_openclose** function does that job and takes care of the file-locking user interface. Despite the nominal contradiction, the file-close operation should be performed in **bt_openclose** as well. We have provided some macros (Listing 6) to simulate separate **bt_open**, **bt_close**, and **bt_extend** functions. This technique can also be extended to our other Btrieve interface routines.

Processing Data Records There are two record-handling functions: **bt_read** and **bt_write** (Listing 7). The function **bt_read** performs any search or get operation that is intended to retrieve a valid data buffer. The key is searched for relative to the value in the key variable (*keyval*). The **bt_write** function must be passed a valid data buffer to perform the update or insert operation.

Using Key Searches The **getkey** key searches (Listing 8) are not intended to return a data buffer. They are simply for the purpose of moving the record position into the position block or establishing whether a specific key exists.

General-Purpose Function Call the **bt_spec** function (Listing 8) for directory and translation operations. When you call Btrieve using these codes, Btrieve needs only a valid position block to perform the operation, except where indicated.

On Your Own

These examples will allow you to quickly add a file-handling library to your system. We have given you Btrieve file-handling structures, code to initialize the parameters for each file, a function to check for memory-resident Btrieve, and the basic function calls needed to intercept Btrieve function error codes.

This programming logic reduces obscure errors by establishing file-handling standards that are reliable and consistent. Murphy's Law as applied to software states that bug-free programs are obsolete. In practice, the rest of the programs (the ones that we are working on, the ones that are most actively used) require constant modification in response to user-requested enhancements. Our goal as maintenance programmers is simple: to modify the program to satisfy user requests in less time than it takes to get in and out of the word processor and recompile the changes (and then take the rest of the day off).

This gives you a platform to customize your file-handling logic to your particular application, taking into account such limitations as memory, storage, and programmer ease of use. PJ

Amrit Khalsa has been a programmer for seven years. By day she is developing a multi-user version of a widely released software package for the state of California; by night she is a small-business consultant.

Pete Maclean specializes as a consultant in computer communications. He lectures internationally on PC networking.

Code follows.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "Btrieve . . ." by Khalsa and Maclean

1

LISTING 1

```
/*   This Btrieve-required structure holds information about one key (or one segment of
 *   a key if the key is segmented). It needs to be defined before the file structure
 *   since it will be included.
 */

struct keyspec      /* key segment structure specification */
{     int  key_pos;    /* start position in record for key segment */
      int  key_len;    /* length of key segment */
      int  key_flag;   /* a vector of bit flags describing
                          attributes of the key */
      long key_tot;    /* unique key count (no dups, nulls)
                          returned by STATUS operation */
      char key_type;   /* a vector of bit flags used to define
                          the data type */
      char null_char;  /* define value of null key (0 or space) */
      char reserve1[4];
};


/*   This Btrieve-required structure holds information about one file. It also includes
 */  the key structure dimensioned to the maximum number of keyed fields within a record.

#define MAX_KEYS  12      /* max indices per file */
#define MAX_KEY_SEGS  22  /* max indices (count segments) per file */

struct filespec               /* file structure specification */
{       int  recl;            /* record length */
        int  page_size;
        int  index_count;     /* number of indexes */
        long tot_recs;        /* total active (non-deleted)
                                 records returned by STATUS operation */
        int  file_flags;      /* a vector of bit flags describing
                                 file attributes (see manual) */
        char reserve2[2];     /* (unused) */
        int  pre_alloc_pages; /* if file is to be pre-allocated assign # of pages */

        /* Note that Btrieve deals with this struct only to this point.
         * We have appended the key structures for our own convenience.
         */

        struct keyspec keybuf[ MAX_KEY_SEGS ]; /* see above */
};
```

LISTING 2

```
struct btfhand
{     char fname[30];          /* file name */
      char pos_blk[128];       /* position block for Btrieve's use */
      long max_var_recl;       /* longest variable recl allowed */
      char owner[25];          /* file owner name */
      int owner_mode;          /* a vector of bit flags used
                                  to set owner lock, data decryption */
};
struct btfkey                    /* key information for one file */
{     int  totsegs;              /* how many key segments in the file */
      int  num_segs[ MAX_KEYS ]; /* number of segments for this key */
      int  start_seg[ MAX_KEYS ];/* starting segment for each key */
      char *name[ MAX_KEYS ];    /* what users call this access path */
};
```

LISTING 3

```
/*   ---------- Btrieve loaded into memory ----------
**    Btrieve puts 033H in interrupt vector 7BH when loaded.

**    Peek at address 7B x 4  = 1ECH
```

2

```
*     ------------------------------------------------------------
*/

void check_bt_loaded()
{
      unsigned resident;

      /* Peek at the interrupt vector */
      if (( resident = peekb(0,0x1EC)) != 0x33)
           puts("MEMORY RESIDENT BTRIEVE NEEDS TO BE LOADED." );
      return;
}
```

LISTING 4

```
void btinit0()       /* initialize TABLE file -- file zero */
{
      /* Initialize Btrieve required Structures */
      btstruct[ 0 ].recl     = 100;          /* init record length */
      btstruct[ 0 ].page_size = 1024;
      btstruct[ 0 ].index_count = 2;         /* not segments */
      btstruct[ 0 ].file_flags = 0;     /* fixed length */

      /* First Index, first segment */
      btstruct[ TABLEF ].keybuf[0].key_pos = 0;
      btstruct[ TABLEF ].keybuf[0].key_len = 2;
      btstruct[ TABLEF ].keybuf[0].key_flag = MOD | SEG | EXT_TYPE ;
      btstruct[ TABLEF ].keybuf[0].key_type = 1;      /* integer */

      /*  key is modifiable, segmented (with segments to follow), */
      /*  extend data type is integer */

      /* First Index, second (and last) segment */
      btstruct[ 0 ].keybuf[1].key_pos  = 50;
      btstruct[ 0 ].keybuf[1].key_len  = 2;
      btstruct[ 0 ].keybuf[1].key_flag = DUP | OK_NULL ;
      btstruct[ 0 ].keybuf[0].key_type = 0;      /* ascii string */
      btstruct[ 0 ].keybuf[1].null_char = 0x20; /* SPACE */

      /*  key has duplicates, null keys allowed */
      /*  null key is defined as space */

      /* Initialize other structures as needed */

      strcpy( fh[0].fname, "TABLE.DAT");  /* assign file name */
      fkey[0].totsegs     = 4;      /* total segments for the file */
      fkey[0].name[0] = "WOOD";           /* name of index 0 */

      /* continue required initialization */
      return;
}
```

LISTING 5

```
/*   Call bt_file() when the file structure is required in the data buffer argument.
 *   Operations:  create(14), stat(15)
 */

int bt_file( opcode, fn )
int opcode;          /* Btrieve operation code */
int fn;              /* file number */
{
struct filespec *fptr = &btstruct[fn];   /* ptr to the file struct */
int buflen=sizeof( struct filespec );
int btstat;
char ext_fname[64];       /* name of file extended across a disk partition */
int in=0;                 /* index */

  if (opcode==B_STAT)  {
     btstat = BTRV(opcode, fh[fn].pos_blk, fptr, &buflen, ext_fname, in);
     if (btstat)  {
          /* error handling */
     }
     else printf("%s%s %ld", "Record count for ",fh[fn].fname, fptr->recl );
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
  }
  else  {
     btstat = BTRV(opcode, fh[fn].pos_blk, fptr, &buflen, fh[fn].fname, in);

     if ( btstat == 20 )  printf("\nBtrieve needs to be loaded.\n");
  }
  return( btstat );
}
/*    Call bt_openclose() to open or close a Btrieve file.
 *    All of the file locking is handled here.
 *        ---file locking---
 *        -1 = accelerated mode
 *        -2 = user has r/o access, others have normal access.
 *        -4 = exclusive lock - no one else can open it
 *'
 *    Operations:  open(0), extend(16), close(1)
 */

int bt_openclose( opcode, fn, lock )
int opcode;          /* Btrieve operation code */
int fn;              /* our file number */
int lock;            /* locking information switch */
{
int buflen,btstat;

     buflen = strlen( fh[fn].owner );

     btstat = BTRV(opcode, fh[fn].pos_blk, fh[fn].owner, &buflen,
                   fh[fn].fname, lock);

     if (btstat)  {
          /* Place error function call here */
     }
     return( btstat );
}
```

LISTING 6

```
#define bt_open(fn,lock)    bt_openclose(0,fn,lock)
#define bt_close(fn)        bt_openclose(1,fn,0)
#define bt_extend(fn)       bt_openclose(16,fn,0)
```

LISTING 7

```
/*   Call bt_read() when data is to be retrieved.
 *
 *   Operations:
 *   get_pos(22)      Get record position # in databuf
 *   get_direct(23)   Get data by sending the record position
 *                    (returned from get_pos()) in data buffer
 *   get_equal(5)     The key buffer must be padded or
 *                    formatted to match exactly.
 *
 *   version(26), step_direct(24), unlock(27)
 *
 *   These operations will return duplicates:
 *   get_first(12), get_next(6), get_prev(7), get_last(13)
 *
 *   These operations won't return duplicates:
 *   get_greater(8), get_less(10), get_gr_eql(9), get_less_eq(11)
 */
int bt_read(op_code, fn, in, databuf, keyval, lock )
int opcode;          /* Btrieve operation code */
int fn;              /* file number */
int in;              /* index number */
char *databuf;       /* data record buffer */
char *keyval;        /* key value */
int lock;            /* application dependant locking # */
{
int buflen;

     opcode += lock;                /* add optional locking */
     buflen = btstruct[fn].recl;
     btstat = BTRV( opcode, fh[fn].pos_blk, databuf, &buflen, keyval, in);
```



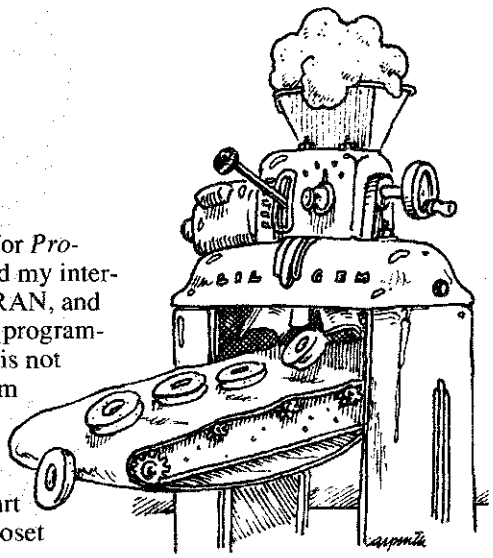
```
     if ( btstat )  {
          /* Place error function call here */
     }
     return( btstat ) ;
}
/*    Pass bt_write() a valid data record.
 *
 *    Operations:  insert(2), update(3), delete(4), unlock(27)
 */

int bt_write( opcode, fn, in, databuf, keyval )
int opcode;          /* Btrieve operation code */
int fn;              /* our file number */
int in;              /* number of index to be used for this operation */
char *databuf;       /* data buffer */
char *keyval;        /* key value */
{
int buflen = btstruct[fn].recl;

     btstat = BTRV(op_code, fh[fn].pos_blk, databuf, &buflen, keyval, in);

     if ( btstat )  {
        /* Place error function call here. */
     }
     return( btstat ) ;
}
/*    Call bt_getkey() for a fast key lookup.
 *
 *    Operations:  Getkey(50+) for opcodes 5-13.
 */
```

LISTING 8

```
int bt_getkey( op_code, fn, in, keyval )
int op_code, fn, in;         /* as before */
char *keyval;
{
int btstat;
int buflen=4;
char dummybuf[4];

     btstat = BTRV(op_code, fh[fn].pos_blk, dummybuf, &buflen, keyval, in);

     if (btstat)  {
               /* Place error function call here. */
     }
     return( btstat ) ;
}
/*   Call bt_spec() as a general purpose function which needs
 *                  and/or returns the key buffer at best.
 *
 *   Operations:  reset(28), stop(25),
 *                begin_trans(19), end_trans(20), abort_trans(21)
 *
 *   get_dir(18)  Directory information is sent or returned
 *   set_dir(17)  in the key buffer for these operations.
 */

int bt_spec( op_code, fn, in, keyval )
int op_code, fn, in;
char *keyval;
{
int buflen=4;
char dummybuf[4];
int btstat;

   btstat = BTRV(op_code, fh[fn].pos_blk, dummybuf, &buflen, keyval, in);
   if (btstat)  {
               /* error handling */
   }
   return( btstat ) ;
}
```


PRODUCED BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Converting Database Files Using Filters



by
M. Steven Baker

Previous articles I have written for *Programmer's Journal* have revealed my interest in number crunching, FORTRAN, and high-performance 80386/680X0 programming. My interest in these areas is not solely intellectual but results from work analyzing the energy use in large buildings with mainframe simulation programs. I'm about to reveal another sordid part of my past and present . . . the closet database programmer.

It all started 18 months ago, when the state agency I worked for got into a jam. One of the department's contracts required that a PC database system on Oregon rivers be developed, and the deadline was less than two months away. The PC database needed to be designed for public distribution and was to be in parallel with a minicomputer GIS (Graphic Information System) mapping system on rivers. As a troubleshooter, I was called in. This was not some modest task but one that required planning, design, a large amount of data-file conversion, and coding a menu-driven user interface in short order.

At the time, I used a prototyper (Bricklin's DEMO program) to develop a fairly complete system demo for management and user approval. On the basis of this experience, I can't recommend prototyping tools enough. Whether it's DEMO II, Proteus, Rapid Prototyping System, or another, these tools allow the simulation of the user interface to an application in weeks without any coding. I would never have met my deadline without this step.

FoxBase+ (dBASE III compatible) was chosen for the target development program because of run-time speed and the unlimited run-time license of less than $300. But the most surprising time consumer was data-file conversion. The raw data files (totaling about 100 megabytes) consisted of a variety of resource data coming from various sources and many mainframe environments, in various flexible and less flexible forms. A large part of my task was file conversion; hence this article.

The most difficult files to convert to a format suitable for importing into dBASE III PLUS (or FoxBase) came from some minicomputer environments. You quickly discover some of the artificial limits in the PC database world, absent from large machines. For example, dBASE III PLUS has an arbitrary limit on record size of 4000 bytes and 128 fields per database file. Sure, we live in a relational world, and one should never create database files this large anyway. Well, try to import a U.S. Geological Service geology database of 6,000 to 8,000 characters per record with variable-length fields, or a hydrosite database with about 400 fields per record received via mag tape as ASCII files—dBASE III PLUS just chokes. I thought surely FoxBase or Paradox would transcend these arbitrary limits, but I was wrong.

My next step was to develop a series of filter programs in C as conversion utilities. I'll illustrate two of the more general ones here.

C Filters for DOS

The first step in developing filter programs is to stop and look around. A most important concept is not to reinvent the wheel. Others have likely also thought about these problems and have developed creative solutions. The UNIX operating system comes with a wealth of text filters, several perfect for database file conversion. One of the more powerful ones is CUT, which allows the selection of fields from a file based on either column position or field delimiters. CUT reads a text file and "cuts" out requested fields to a destination file (the default is stdout). Not having a UNIX system in house, I thought the MKS toolkit (from Mortice Kerns) would provide a suitable DOS version of CUT. But alas, the MKS cut implementation has a line-length limit of 1024 characters. So I wrote a quick-and-dirty DOS version with more-generous limits.

I use a standard template for writing filters in C, which allows me to reuse about half the code from one application to another. These file filters consists of three basic parts: (1) the main routine that processes the command line arguments; (2) a routine to open input and output files (if need be); and (3) the actual file filter. Every attempt is made to keep the code as portable as possible. I port these same utilities to a Prime minicomputer, which has—How shall I say it?—a less-than-ideal C compiler. Very often it is more convenient to conduct the filtering on the minicomputer (assuming a C compiler is available), saving PC disk space and time.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

For processing command line arguments in a consistent way, I use the AT&T **getopt** routine. My understanding is that AT&T placed the C code for **getopt** into the public domain to facilitate command line uniformity. (Note that AT&T neglected to put the **getopt** manual page in the public domain.) A compatible version of **getopt** is supplied with Borland's Turbo C, which may also be used with Microsoft or WATCOM C compilers with a few minor changes. The standard AT&T version works just fine with most C compilers.

Again emulating previous successful designs whenever possible, most of the UNIX options are implemented so that the program's behavior would be similar to its model. An output file option (*-o*) has been added for convenience (a necessity for the Prime, which doesn't support simple file redirection). CUT supports selecting individual fields separated by commas or in combination with ranges, such as 103–156. The DOS 128-character command line is the only limiting factor. For cutting by fields, the default delimiter character is the tab, but it can be changed with the *-d* option. The following example would be used to cut columns 1 through 10, 20, 30, and 80 through 120 from the infile:

```
cut -c"1-10,20,30,80-120" -o outfile infile
```

Importing Data Files

Cutting by columns works well with a certain class of data files—those with fixed-length fields and records. Most mainframe data-file managers using SQL will generate reports with a fixed column format. CUT works well in removing only those data fields required and skipping over blank columns in an SQL report. I've also used grep to remove the page headings and blank lines from these SQL reports before importing. On fixed-length files of reasonable record length (that is, less than the magic 4000-character dBASE III limit), this task can be accomplished using dBASE itself. But I've found that using CUT is easier, it saves on disk space, and it can be built into a simple batch file for regular data translation.

The second class of foreign data files—those with variable-length fields in a record—is more difficult: It requires cutting by fields. Fortunately, most database software in this category (such as the minicomputer precursor to Revelation) can usually dump an ASCII text file with fields delimited by some user-chosen character, such as # or ~, that isn't present in the data file. The second utility, DELIM, handles some additional translation problems with these variable-length fields.

The C code for CUT is downright ordinary, illustrating the virtue of using the UNIX utility as a model. The maximum line length is set with a **#define** for easy customization. The current version has MAXLINE defined as 8192, which should handle most of your data files. In parallel with this line buffer is a flag array **map,** also of MAXLINE bytes, that is used to indicate which fields are to be cut. A **char** array was used for this logical flag table for simplicity, with 0 meaning false (no cut) and 1 meaning true. If you ever need to process files with 30–50K line lengths, the map array could be reduced by a factor of 8 using the 8 bit fields in each byte.

Ironically, most of the program went into coding the **setfield** routine called by main to read the command line and set the bytes in the map array to those fields or columns to cut. I used a simple finite-state machine of four states: BEGIN, NUMBER, COMMA, and RANGE, depending on the last character. The actual conversion routine is very simple, supporting the two options—column or field cutting. The **stoi** routine is a replacement for **atoi**, modified from a version by Allen Holub in *Dr. Dobb's* many years back. One point on which I always disagree with the UNIX philosophy is help. If CUT is invoked without any arguments, detailed help for proper usage is printed to stderr. This mitigates the necessity of manual pages for all these simple programs.

The second utility, DELIM, is very similar in structure. Its purpose is to convert an ASCII text file with fields separated by a unique delimiter character to a dBASE III–compatible comma-delimited file. DELIM must add commas and quotes around fields for dBASE compatibility and to preserve spaces and commas in fields. When exporting a mainframe database to ASCII, if any database character fields contain comma characters, then the comma cannot be used as the field delimiter. An occasional quote character in a field will also play havoc with the dBASE **append** command. DELIM allows the user to choose a delimiter character (tab is the default) and will also replace any double quote (") character found in data fields with some other character. The period (.) is used to select a space for the replacement quote character. Two of the routines, **dofiles** and **usage**, are so similar to CUT that they haven't been included. Again, the actual filter uses another finite-state machine using two variables: *state* for the current state and *last* for the previous state. Most of one's debugging is spent getting all the possible end-point conditions covered. This task would be an ideal application for a logic compiler like LogicGEM from Sterling Castle.

C is an ideal language for writing quick filters, a benefit of a standard command line interface and stdio library calls. C also provides reasonable portability between different environments—much more so than Pascal or BASIC. If one wanted real speed, these programs should be revised to read the input and write the output in big chunks and provide their own buffered I/O routines. These simple pragmatic conversion tools can provide you with a template for many more filters. The one limit of this simple design remains the DOS 128-character command line length (most UNIX versions support at least a 4096 character command line). Adding a *-f* option to read the command line from a file is left as an exercise for those who need it. Another useful filter I haven't written yet is the ancillary to CUT—PASTE. I guess I haven't needed it badly enough. Of course, pasting can be accomplished using a simple dBASE or FoxBase script. But maybe someone else will take up the challenge. PJ

Steve Baker is the editor of Programmer's Journal.

Code follows.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "Converting . . ." by M. Steven Baker

1

LISTING 1

```
/*  cut.c    - a unix like filter
 *
 *  Author:                 M. Steven Baker
 *  Last Revision:          October 17, 1988
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

#ifndef FILENAME_MAX
#define FILENAME_MAX     68
#endif

FILE *infp;
FILE *outfp = stdout;                   /* our file pointers */

char  infile[FILENAME_MAX] = "";    /* our file names */
char outfile[FILENAME_MAX] = "";

int verbose = 0;
int errflag = 0;

char delim = '\t';              /* default delimiter character */
int fieldflag = 0;              /* logical flag for field delimited cut */
int colflag = 0;                /* logical flag for column oriented cut */
int sflag = 0;                  /* the cut -s flag -- not implemented yet */

char *pgm = "cut";          /* our program name */

#define MAXLINE     8192

char map[MAXLINE+2];            /* our field map array */
char line [MAXLINE+2];          /* our fgets() line buffer */

void usage();

char *help_msg[] = {
     "CUT various columns or fields from an ascii text file",
     "The following options may be used singly or in combination:",
     "-c columns \t cut columns (ie 1,2,5,7-20)",
     "-d delimiter \t field delimiter character (default = tab)",
     "-f fields \t cut fields (ie 1,2,5,12-30)",
     "-o filename\t create output text file (default stdout)",
/*   "-s \t\t don't copy lines without delimiters" , */
     "-v\t\t verbose mode"
};

main(argc,argv)
int argc;
char **argv;
{
     int  ch;                       /* variable for getopt */
     int i;

     extern int getopt();
     extern char *optarg;
     extern int optind, opterr;

     for (i = 0; i < MAXLINE; i++)      map[i] = 0;          /* zero map array */

     /* process command-line arguments */
     opterr = 0;     /* first turn off error messages from getopt */
     while ((ch = getopt(argc,argv, "vsc:d:f:o:")) != EOF) {
          switch (ch) {
```

2

```
               case 'c':
                    colflag = 1;
                    if (setfields(optarg) == EOF) {
                         errflag++;
                         if (verbose)
                              fprintf(stderr,"%s: Error setting columns to cut\n",pgm);
                    }
                    break;
               case 'd':
                    delim = *optarg;
                    break;
               case 'f':
                    fieldflag = 1;
                    if (setfields(optarg) == EOF) {
                         errflag++;
                         if (verbose)
                              fprintf(stderr,"%s: Error setting fields to cut\n",pgm);
                    }
                    break;
               case 'v':
                    verbose = 1;
                    break;
               case 'o':       /* output specifed cut text file */
                    strcpy(outfile, optarg);
                    outfp = NULL;
                    break;
               case 's':       /* not implemented */
                    sflag = 1;
                    break;
               default:        /* unknown option */
                    errflag++;
                    break;
               }
          }
     if (colflag && fieldflag) errflag++;     /* can't have both options */

     if (errflag || !(argc-optind) ) {
          usage(pgm);
          exit(1);
     }
     if ( dofiles(argc - optind, argv += optind) )     /* did we have errors? */
          exit (1);

     exit(0);
}

/* cut -  process each filename or standard input */

int dofiles(ac, av)
int  ac;
char **av;
{
     int  ch, errcount = 0;
     char *p;

     for (; ac > 0; --ac, ++av) {
          p = *av;
          strcpy(infile,p);              /* store binary data filename */
          if ((infp = fopen (infile,"rt")) == NULL) {
               fprintf(stderr,"%s: ERROR opening input file: %s\n",
               pgm,infile);
               continue;
          }
          if (outfp != stdout) {
               if ((outfp = fopen (outfile,"wt")) == NULL) {
                    fprintf(stderr,"%s: ERROR opening output file: %s\n",pgm,outfile);
                    fclose (infp);
                    continue;
               }
          }
          if (cut(infp,outfp)== EOF) {
               fprintf(stderr,"%s: Error converting file: %s\n",pgm,infile);
               fclose(infp);
               fclose(outfp);
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
                errcount++;
                continue;
           }
           fclose(infp);
           if (fclose(outfp) == EOF) {
                fprintf(stderr,"%s: Error closing output file: %s\n",pgm,outfile);
                errcount++;
                continue;
           }
      }                         /* end of for loop */
      return (errcount);
}
/* do the actual cut */
cut(fp1,fp2)
FILE *fp1, *fp2;
{
      int i;
      char c;
      char *p;

      while ((p = fgets(line, MAXLINE+2, fp1)) != NULL) {
           i = 0;
           for (c = *p; *p; c = *++p) {
                if (fieldflag) {
                     if (map[i] || c == '\n') fputc(c,fp2);
                     if (c == delim) i++;
                }
                else {
                     if (colflag) {
                          if (map[i++] || c == '\n') fputc(c,fp2);
                     }
                     else fputc(c,fp2);
                }
           }
      }
      return (0);
}

#define BEGIN  0
#define NUMBER 1
#define COMMA  2
#define RANGE  3

/* setfields - set fields to cut from command line */

int setfields(optarg)
char *optarg;
{
      int i, last, state;
      char c;
      char *p;

      i = last = state = BEGIN;
      p = optarg;

      if ((c =*p) == '\0') return (EOF);

      while ((c = *p) && i < MAXLINE) {
           switch (state) {
                case BEGIN:
                     i =stoi(&p);                    /* now pointing to next char */
                     if (i > 0 && i < MAXLINE) {
                          last = i;
                          map[i-1] =1;
                          state = NUMBER;
                     }
                     else return (EOF);
                     break;
                case NUMBER:                    /* a number was last */
                     if ( c == '-')
                          state = RANGE;
                     else {
                          if (c == ',')  state = COMMA;
                          else return (EOF);
```



```
                     }
                     p++;                       /* increment pointer */
                     break;
                case COMMA:
                     if (isdigit(c)) {
                          i = stoi(&p);         /* stoi increments pointer */
                          if (i > 0 && i < MAXLINE) {
                               last = i;
                               map[i-1] = 1;
                               state = NUMBER;
                          }
                          else return (EOF);
                     }
                     else return (EOF);
                     break;
                case RANGE:
                     if (isdigit(c)) {
                          i = stoi(&p);             /* stoi increments pointer */
                          if (i > 0 && i < MAXLINE) {
                               if (last < i) {
                                    while (last < i)
                                         map[++last -1] = 1;
                                    state = NUMBER;
                               }
                               else return (EOF);
                          }
                          else return (EOF);
                     }
                     else return (EOF);
                     break;
                default:
                     return (EOF);
           }
      }
      return (0);
}

/* STOI.C More powerful version of atoi chopped from the Allen Holub stoi(c) */

int stoi(instr)
char **instr;
{
      /*   Convert string to integer. Conversion stops on encountering
       *   the first character which is not a digit.
       *   *instr is updated to point past the end of the number.

       *  patched to ignore leading '-' sign, since we need this for range checking
       */

      int  num  = 0 ;
      char *str = *instr;

      while( isspace(*str) )
           str++ ;
      while( '0' <= *str  &&  *str <= '9' ) {
           num *= 10;
           num += *str++ - '0' ;
      }
      *instr = str;
      return( num );
}

/* usage - display an abbreviated help usage message */

void usage(pname)
char *pname;
{
      int i, n = sizeof(help_msg)/sizeof (char *);

      fprintf(stderr,"Usage: %s [options] file...\n\n",pname);
      for (i = 0; i < n; ++i) fprintf(stderr, "%s\n",help_msg[i]);
      return;
}
```

Code continues . . .

1989 Programmer's Journal 7.3
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Baker code, continued

5

LISTING 2

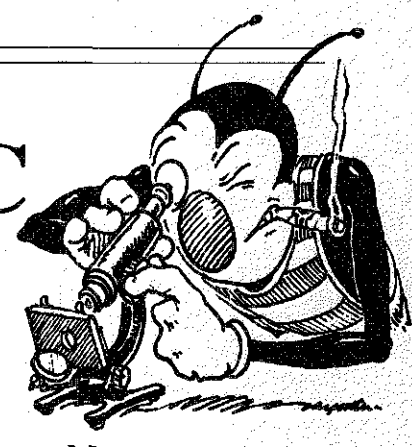
```
/*  extracts from delim.c    - a dbASE3 filter
 *
 *  takes a text file with fields delimited with a 'delim' char
 *  and creates a file a comma delimited file with double quotes '"'
 *  around all fields so dBASE3 can import it
 */

/* noquote is a boolean variable set to 1 if double quote char's in fields are to be
 * replaced with the 'new_quote' char variable which by default is a space
 * character */

#define BEGIN    0
#define INFIELD  1
#define DELIM    2

/* procedure delim -- copy file to file replacing delimeters */

delim(fp1,fp2)
FILE *fp1, *fp2;          /* the input and output file pointers */
{
    int c;                /* 1 char buffer */
    int i,state, last;

    long count = 0;       /* count of characters processed */
    long linecnt = 0;     /* number of lines processed */
    long quotecnt = 0;    /* count of quote characters replaced */

    state = BEGIN;
    while( (c = getc(fp1)) != EOF) {
        count++;
        switch (state) {
            case BEGIN:
                if (c == delim) {
                    fputc('"',fp2);        /* emit the first " */
                    fputc('"',fp2);
                    state = DELIM;
                    last = DELIM;
                }
                else {
                    if (c == '\n') {
                        fputc (c, fp2);
                        state = BEGIN;
                        linecnt++;
                    }
                    else {                  /* got a starting CHAR */
                        fputc('"',fp2);     /* emit the first " */
                        if (noquote) {
                            if ( c == '"')  {
                                c = new_quote;
                                quotecnt++;
                            }
                        }
                        fputc (c, fp2);
                        state = INFIELD;
                    }
                }
                break;
            case INFIELD:
                if (c == delim) {
                    fputc('"',fp2);        /* emit the first " */
                    state = DELIM;
                    last = DELIM;
                }
                else {
                    if (c == '\n') {
                        fputc ('"',fp2);
                        fputc (c, fp2);
                        state = BEGIN;
                        linecnt++;
                    }
                    else {                  /* got a CHAR */
                        if (noquote) {
```

6

```
                            if ( c == '"')  {
                                c = new_quote;
                                quotecnt++;
                            }
                        }
                        fputc (c, fp2);
                    }
                }
                break;
            case DELIM:
                if (c == delim) {
                    if (last == DELIM) {
                        fputc(',',fp2);
                        fputc('"',fp2);
                        fputc('"',fp2);
                    }
                    else {
                        fputc('"',fp2);     /* then emit the first " */
                        state = DELIM;
                        last = DELIM;
                    }
                }
                else {
                    if (c == '\n') {
                        fputc (c, fp2);
                        state = BEGIN;
                        linecnt++;
                    }
                    else {                  /* got a CHAR */
                        fputc (',',fp2);
                        fputc ('"',fp2);
                        if (noquote) {
                            if ( c == '"')  {
                                c = new_quote;
                                quotecnt++;
                            }
                        }
                        fputc (c, fp2);
                        state = INFIELD;
                    }
                }
                break;
            default:
                return (EOF);
        }
    }
    if (verbose) {
        printf("\n  Characters processed: %ld   Lines processed %ld\n",count,linecnt);
        if (noquote && quotecnt  0)
            printf("  Quote characters replaced: %ld\n",quotecnt);
    }
    return (0);
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The Proposed ANSI C Language Standard



by
Rex Jaeschke

© *Rex Jaeschke, 1989. All rights reserved.*

This month we'll revisit a topic I covered last month. We'll also take a quick look at WATCOM's latest effort and a more detailed look at Ecosoft's ECO C.

Structure Expressions Revisited

In my last column, I showed an example of four different structure expressions. I pointed out that some DOS C compilers couldn't handle them and that some of the vendors of those products agreed there was a problem. However, after much discussion, I find that I was wrong—as were numerous others. Since the explanation is subtle, I thought I would revisit those expressions with the "real" answer. The example was something like

```
struct s {
    char m[30];
};

main()
{
    struct s frs(void);
    char *pc;

/*1*/  pc = (s1, s2).m;
/*2*/  pc = frs().m;
/*3*/  pc = (5 ? s2 : frs()).m;
/*4*/  pc = (s1 = s2).m;
}
```

Contrary to my earlier claim, all four cases are incorrect. According to ANSI C, "an *lvalue* is an expression (with an object type or incomplete type other than void) that designates an object." Now, we all know that expressions that designate arrays are converted to pointers to the first element. However, there are exceptions, and they involve the **sizeof** and **&** operators. That is, in the expressions *sizeof(array)* and *&array*, the expression array is *not* converted to a pointer. (The idea of a pointer to an array is not new, although allowing *&array* is—*&array* has quite a different meaning than *&array*[0], even though they result in the same address.)

ANSI C further states, "Except when it is the operand of the **sizeof** operator or the unary **&** operator, or is a character string literal used to initialize an array of character type, . . . an *lvalue* that has type *array of type* is converted to an expression that has type *pointer to type* that points to the initial element of the array object and is not an *lvalue*."

What this means is that the only time an array expression is required to be converted to a pointer is when it is an *lvalue* and it is not in the presence of the operators specifically mentioned. In other words, if a non-*lvalue* expression (formerly referred to as an *rvalue*) has an array type, the behavior is unspecified in regard to conversion; that is, the conversion is not required.

This is most important, since the array expression in cases 1–4 in our example are not *lvalues* and therefore are not required to be converted. Regarding the dot operator, ANSI C states, "The value [of a dot expression] is that of the named member, and is an *lvalue* if the first expression is an *lvalue*." By definition, the comma operator, function call, ternary operator, and assignment do not result in an *lvalue*. Therefore, such expression followed by a dot operator are not *lvalues* and are not required to be converted to pointers.

What, then, is the utility of such expressions? As far as I can see, there is absolutely none, since you can't do anything with an array except take its size. All other operations on array expressions rely on the array or pointer conversion taking place.

WATCOM C V7.0

It was June 1988 when I reviewed WATCOM's first DOS release, V6. Release V6.5 came out several months later, and now V7 is shipping. According to the vendor, this *is* an ANSI release.

As a technician, I always love a challenge to "prove me wrong." I ran their product through my battery of tests and specifically looked at the shortcomings I mentioned the last time around. (You may have found that I review products unevenly. This is by design: The closer a product claims to be in conformance to the standard, the more critical I become in detailing discrepancies.) And now, the envelope please . . .

WATCOM V7 is not ANSI conforming, but by the tests I ran, it's better than 99 percent there. The outstanding problems are relatively minor, and you can fix almost all of them by editing the headers yourself.

The **assert** macro in **assert.h** is not defined correctly when **NDEBUG** is defined. It must be defined as a void expression, while it currently gets defined as nothing. You cannot use **assert** as the first operand to a comma operator or as the second or third operands of a **?:** operator.

If you define the macro **NO_EXT_KEYS**, you are presumably getting an ANSI-only compilation environment. However, several nonstandard identifiers are defined outside this conditional envelope in various headers.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

QEdit Advanced

the "quick editor"

By SEMWARE

QEdit is one of the smallest, fastest, most easy to use multi-file editors available for IBM Compatible Computers with many features to enhance your productivity.

- *Extremely Fast*
- *Completely Configurable (Including Keyboard!)*
- *Easy to Use Macro Facility*
- *Optional "Pop Down" Menu System*
- *Recover Deleted Text*
- *Edit Dozens of Files Simultaneously*
- *Utilize up to 8 Windows to View Files*

QEdit *Advanced* requires an IBM PC/XT/AT or Compatible, IBM PS/2 30-80, 128KB of available memory. Please specify 5.25" or 3.5" diskette.

$54.95 MASTERCARD & VISA ACCEPTED

Add $3.00 Shipping & Handling. Georgia Residents Include 4% Sales Tax.

To Order:
SEMWARE, 730 Elk Cove Court
Kennesaw, Georgia 30144
Phone Orders: (404) 428-6416

Number 148 on the Reader Service Card

The **va_end** macro in **stdarg.h** is required to be a void expression. Currently it is not.

Several standard headers **#include** other standard headers, and I don't believe this is workable. The problem is that file-scope names such as **offsetof** (a macro in **stddef.h**) are defined *only* if their parent headers are included directly by the programmer; they are not universally reserved names (such as **fopen**, for example). Therefore, when I include **time.h**, I don't expect to find—and should not find—**offsetof** defined as a side effect (as it is in WATCOM V7).

All the problems that I listed in my review last year have been fixed, except one: They still incorrectly permit floating-point constants in **#if** arithmetic. They have even added **CLOCKS_PER_SEC**, a macro in **time.h** invented near the end of 1988, after the ANSI C meeting. Except for the nitpicking exceptions discussed above, WATCOM's claim that they reflect the current standard state is justified.

There has been a lot of discussion about how WATCOM may well have the fastest and most compact code generation in the DOS C world. Certainly, it has been my experience that they are leaders rather than followers. For ANSI C conformance, WATCOM V6.0 set a standard no one else has beaten, and now, with V7, they have raised that standard even higher. If you are serious about developing in modern C under DOS, WATCOM V7 is a product that warrants your attention.

ECO C V4.18

This month we turn the microscope on ECO C, from Ecosoft. Founded by Dr. Jack Purdum, a well-known C author and teacher, the company and the ECO C compiler has been around for quite a spell and has a dedicated following.

To be fair, Ecosoft does not claim their compiler is ANSI standard, nor do they claim it is even close to it. Chapter 3 of the manual discusses the ANSI enhancements supported in this release and provides a minitutorial on their use.

The Compiler Options

There's very little to say here except that I disabled nested comments and selected "pick" level 9. You can select a picky level up to 9, where each higher level does more quality checking. The default level is quite low; anyone doing serious development and not having to rework older code should use at least level 7.

Many of the messages produced are very descriptive and useful. However, on numerous occasions, I found the error-recovery mechanism to be wanting. The compiler sometimes tried to recover from an error and continue, thus generating many more messages; at other times, it didn't like what it saw and stopped compiling altogether. For the first time in using my evaluation suite, I had to rearrange quite a few tests to get the information I wanted.

A major omission, from my perspective, is that there is no option to look at the output generated by the preprocessor.

The Preprocessor

The following ANSI C attributes are not supported or have some limitation:

- There is no #, ##, or **defined** operator.
- Four of the five predefined macros are provided, yet they are not seen by the **#ifdef** and **#ifndef** directives as macros.
- Macro definitions containing their own names are recursive.
- Extra tokens at the end of directive lines are not diagnosed.
- **#error** is not implemented.
- A macro cannot be defined more than once, even with exactly the same token sequence.
- **#undef**ing a nonexistent macro is considered an error.
- Long **int** constants in **#if** arithmetic are not liked.
- In **#if** (int), the preprocessor complained about the keyword **int** when it should have treated it as an undefined macro with value 0.
- Floating-point constants in **#if** arithmetic should not be permitted.
- There is no support for the **#include** identifier directive format.

In general, I found that parts of the grammar used by the preprocessor were not correct even for K&R. The preprocessor expected tokens to be in very specific places, and it ignored excess tokens if it deemed they shouldn't really be there. Also, several preprocessor tests resulted in garbage error messages, presumably due to a stray pointer into memory.

Several **pragma**s are defined for DOS-specific code generation. A really great feature is the warning message produced when an unknown **pragma** directive is encountered. (This is not

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

required by ANSI C, since unrecognized **pragmas** are supposed to be ignored.)

The **sizeof** operator is permitted in **#if** arithmetic, although this is not required by ANSI C.

The C Language

The following ANSI features are not supported or have problems:

- The general backslash token continuation feature is not implemented.
- There can be no bit fields in unions unless they are in structures first.
- No auto-aggregate initializers are permitted.
- There is no support for **const**, **volatile**, or **signed** keywords.
- There is no f (or F) **float** constant suffix.
- There is no **long double** data type.
- Both "U" and "L" suffixes are permitted, but "UL" is not.
- There are no trigraphs.

For the most part, function prototypes seem to work fine; however, there were a few problems. The compiler did not like a prototype to be specified in a function pointer declaration. Also, if you define a function with a formal argument list containing the ellipsis punctuator, an error results. This punctuator is permitted in a function declaration, however.

The Run-Time Library

The following standard headers are missing: **float**, **limits**, **locale**, **signal**, **stdarg**, and **stddef**. Note though, that **stdarg**'s predecessor, **varargs**, is provided. The **mem*** functions are in the nonstandard header **memory.h**, rather than in **string.h**. Similarly, a few other identifiers are in nonstandard places.

Without exception, all the standard header files provided need work to approach ANSI conformance, with some needing major surgery. Because of this, I see no point in producing a detailed list of the discrepancies.

The Bottom Line

While present customers will no doubt be happy with the enhancements provided with this release, this product is not in the mainstream race to adopt the ANSI standard. However, the picky error checking available and the relatively low cost of the product could make it useful as an auxiliary tool through which to pass your code when getting a second opinion.

Flash!

The ANSI X3 Committee on March 3 approved the proposed C standard and forwarded it to ANSI for final administrative review. At long last, we may have a final, approved ANSI C standard. PJ

Rex Jaeschke is an independent computer consultant, author, and lecturer. He is a member of the X3J11 C Standards Committee, an observer on the ISO C Standards Committee, founding editor of The C Journal, *and the C language editor of* DEC Professional. *His new book,* Portability and the C Language, *was published by Hayden Books in October 1988.*

If you would like to comment on or question any aspect of the ANSI C Standard, contact Rex at 2051 Swans Neck Way, Reston, VA 22091 or via uucp: uunet!aussie!rex. *Rex will endeavor to address your concerns in future columns as space permits.*

SuperFAST!

C-Index™

Microsoft C Turbo C

Unix VAX Mac Windows OS/2

Database Toolkit

Nine simple function calls give you the power of fast B+Tree indexing and automatic variable length records.

C-Index makes data management programming easy with a friendly Application Program Interface, excellent documentation, extensive examples and a tutorial database application. C-Index is ready-to-use with Microsoft C or Turbo C, right out of the box. No need to compile or change the source code.

C-Index is the fastest database toolkit available, up to ten times faster than C-Tree and CBTree! C-Index is also the most flexible package you can buy with 9 high level functions and 37 building-block functions that allow you to tailor C-Index to every program.

C-Index is the library of choice for thousands of software developers. C-Index has been successfully used in mass market commercial applications, such as **Prime Time,** the highly acclaimed personal time manager. Wayne Ratliff, author of dBase II and dBase III, thought C-Index was so "terrific" that he uses it as the foundation for his new **Emerald Bay** database system.

Discover for yourself the power of C-Index. Call to order your own copy today.

- No Application Royalties
- Complete Source Code
- Single and Multi-user Access
- 170 page manual
- Proven Reliability
- Fast B+Tree Indexing
- Variable Length Records
- Random and Sequential Access
- Data and Indexes in same file
- Multiple Record Formats per file

"I heartily recommend this package."

Dr. Dobb's Examining Room, June '88

C-Index™ for Turbo C® **$99.00**

Complete source and libraries for Turbo C 1.5 and 2.0

C-Index™ for Microsoft C® **$195.00**

Complete source and libraries for Microsoft C versions 5.0 and above.

C-Index/Plus for Unix/Xenix and All Other Systems **$395.00**

Includes both above libraries plus all source code and test routines required to port to any system: Unix, VAX, Mac, Windows, DOS and OS/2.

Trio Systems **213/394-0796**

2210 Wilshire Bl. Suite 289 Santa Monica, CA 90403

Number 262 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A Substitute for the Missing TYPE Command in Windows

by
William S. Hall

Technically speaking, you don't need this program. If you want to look at a file while running Windows, you can employ a variety of means to do so. If the file is less than 16K, you can use Notepad. If it is larger, you can try Write or some of the new Windows word processors, such as AMI. You can even run COMMAND.COM from Windows and then use the DOS **TYPE** command. However, each of these methods use a fair amount of memory, and only COMMAND.COM affords a display that scrolls quickly through the entire file. This is often all you need to ascertain its contents.

So, I have written Winvue, a simple Windows utility that displays any file in much the same manner as the MS-DOS **TYPE** command. Not only is the program useful, but it is also especially interesting to the programmer because of the techniques employed to read and process Windows messages. As you may know, the main loop in a Windows program is one that requests and dispatches messages. Windows then calls the program's window procedure with the message, and while the message is being acted upon, no other Windows program can run. Some mechanism is necessary to share the CPU with other programs while the program carries out an extended process. You will see how this can be done in Winvue using function pointers and two different ways of reading messages.

If you are new to Windows, you may want to refer to some of our previous articles in *PJ* or look at a book or two on the subject (see references). You are probably aware that the mere task of getting a window to display and maintain itself involves a fair amount of boilerplate; architectural issues, such as segment organization, are also involved. We have already covered these topics in some detail. Here we are going to concentrate on new topics, such as using **PeekMessage**, creating menu accelerators, and opening files.

Regardless of your background, if you are interested in Windows or Presentation Manager (PM), do read on. Rather than discuss all the aspects of every module, I shall try to keep the discussion as self-contained as possible and supply appropriate references when needed.

The Disk Offering

Although the main body of Winvue is fairly small, one routine—selecting and opening the file to be displayed—makes it impractical to present all of the source code in the article. So, we are going to follow our usual practice of providing only excerpts and asking you to purchase *PJ*'s disk for this issue or go through one of the bulletin board services to obtain the complete source code along with the executable file.

To display text in the client area, Winvue uses the scrolling window code development shared by programs Winaux and PMaux, which were presented in *PJ* 7.2. For Winvue, it was necessary to include some additional features, such as the ability to clear the screen and to switch between fonts. So, I have taken the opportunity to provide a Clear Screen option in PMaux and Winaux. Along the way, I have added logging capability and better initial positioning code to Winaux. The code disk for this issue of *PJ* includes new versions of these two programs.

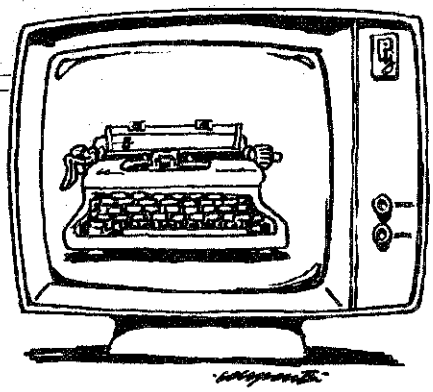
Using Winvue

Winvue is very simple to use. To display a file, choose "View" from the Files menu (or use the accelerator key, F1) to generate a standard file-selection dialog box. When you select a file, its contents will begin to scroll upward from the bottom of the Winvue window. You can suspend the display with CTRL-S or the "Pause" menu item. Pressing CTRL-Q or selecting the "Resume" menu selection will cause the scrolling to continue. The left mouse button also acts as a pause/resume toggle. Finally, you can cancel the action by using the corresponding menu item or pressing CTRL-C.

Although the file selection defaults to an extension of TXT, files of any type can be viewed. A simple filter allowing masking of the eighth bit of each character is an option. More complex options for the future could include conversion mechanisms for certain word processor files or a dump-style display for binary files. Another interesting extension would be a transparent display to suppress the action of most control characters and allow their character representations to be viewed.

The maximum number of lines that can be seen at one time depends on your display type and is optimized for the size of a maximized window. Once a line scrolls off the top of the maximized display, it is lost. A useful program addition would buffer more lines and provide scroll bars to move back to lines that have already gone by.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



If you use Notepad or some other text-oriented program in Windows to view your file, you can use the Clipboard to move text to another application. This is true even if you simply type a file using COMMAND.COM in a window. Winvue does not currently have this capability, but in a future article, I will try to show how to add this and some of the other features mentioned above.

Winvue allows one file to be specified on the command line; so, if you execute the program using the Windows "Run" option and supply a filename, it will load and display that file.

Font Selection

Before viewing a file, you can choose one of two fonts for the display: the Windows System font or the Windows Terminal font. The default is the System font, which is the one you see in menus, dialog boxes, window titles, and the client area in some Windows applications, such as Notepad. This font uses a modern typeface and is always available. It comes in one size, which depends on the display. On the EGA, the System font is 8 bits wide and 10 bits high (counting external leading). For the VGA, the System font is 14 bits high and 8 bits wide. On the IBM 8514/A, the System font is larger yet. The character set is ANSI. Characters from 32 to 127 are standard ASCII, and those in the range 128 to 255 contain the additional letters and symbols needed in major Western European languages. There are no symbols for the control characters. You can see any character in the font from Notepad by holding down the ALT key while typing "0*nnn*" on the number pad, where *nnn* is any number between 32 and 255. Thus, 065 will display the letter *A*, and 161 will show the inverted exclamation point (¡) used in Spanish.

Winvue also allows you to view your file in the Windows Terminal font, which is sometimes called the OEM (original-equipment manufacturer) font, because it is supposed to be similar to the machine's ROM font. The Terminal font is normally used only when running a standard application in a Window. For example, if you execute COMMAND.COM from Windows, the Terminal font is the one you see in the client area. You can also use some of these characters in certain Windows applications by typing ALT-*nnn* (omitting the 0 before the three-digit sequence). Windows will then convert the OEM character at position *nnn* to the corresponding ANSI one or, if there is no equivalent, to a +.

Of course, files created by DOS do not use the Windows System font. If you had a DOS file in a language using letters requiring diacritical marks or some of the graphics symbols (which are in the 128 to 255 range), for example, and wanted to see them as they were created, then you would select the Terminal font option when using Winvue. To illustrate the differences, we have supplied file CSET.TXT on the disk. Look at this file in Winvue first using the System font and then using the Terminal font to see some of the differences between the two character sets. What will not be apparent from this simple test is that the Terminal font also has symbols for most of the characters in the 0 to 31 range. (As an exercise, try to add to Winvue the transparent option mentioned above, and send me a copy.)

Winvue's Organization

Winvue is written in C, using the medium memory model. There are five compilation modules, which at run time live in three code segments. One contains code used only to initialize the program; it is marked in the definition file as PRELOAD, MOVEABLE, and DISCARDABLE. A resident segment contains the program's entry point **WinMain**, the Windows procedure and its various support functions, and the code needed to display the text. It is marked as PRELOAD and MOVEABLE. Finally, there is a separate segment containing the code needed to list and select files for display. It is marked as LOADONCALL, MOVEABLE, and DISCARDABLE. You may want to see my articles in *PJ* 6.4 and 7.2 for examples of typical make and definition files to see how segments can be named and assigned attributes in Windows.

With this organization, 2K of initialization code can be discarded by Windows after Winvue is loaded. Another 2K for managing the file-open dialog box is loaded only when needed and is discarded when finished with. To see the effect of this kind of segmentation, strike the F1 accelerator key after loading Winvue, and watch your hard disk light while Windows loads the segment needed to display the file list. If you again use F1 after the file is displayed, no disk activity will take place, since that segment is loaded (unless some other Windows activity has caused it to be discarded).

An even more optimum arrangement would be to place the text display code in its own segment and to move the text display initialization code, which is currently in the TTYCLS.C module, into the initialization segment. In this way, you can further reduce the resident code. In particular, Winvue can be maintained as an icon with just enough resident code to sustain basic Windows operations without constantly reloading segments. This design is important in Windows, because there is simply not a lot of run-time memory available.

Resources in Winvue

Listing 1 shows an excerpt of the resource file for Winvue. At the bottom, you see the description of the dialog box that will appear when a file is opened for display. It is now standard in Windows to use this format. Filenames, paths, and search specifications can be entered in an edit box. A list box shows all files in the current directory that match the search specification. Two pushbuttons allow selecting a file or canceling the operation. Finally, there are two static fields. One simply names the general contents of the edit box; the other, which initially contained a null string, displays the current path.

A number of special calls are available for managing this dialog box. Nevertheless, it is some of the most annoying

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

code to write in Windows. I adapted the version in Winvue using the Microsoft-supplied file DLGOPEN.C from the sample source code in the Windows Software Development Kit. In its original version, DLGOPEN.C is 350 lines of C code. Ours has been reduced to a mere 296 lines. Not only is it tiresome to write or modify, but similar code is repeated in every other Windows application requiring file services. If you are running five different Windows programs simultaneously, you may have code that accomplishes the same task loaded five times in memory, unless the authors have been careful to put the file-open manager into a discardable segment. My mouse for a well-written dynamic link library performing this function!

Accelerators come in two flavors in Windows. First, it may help to know that everything can be done in Windows without a pointing device (i.e., mouse). Some applications are even written deliberately to depend entirely on the keyboard. Merrill Lynch is supposed to have developed a Windows program for its stock traders, whose desks, it is reputed, are simply too cluttered to have any additional hardware on them. (Why not a touch screen? I once wrote a combination mouse–and–touch-screen driver for Windows, but AT&T decided not to manufacture the hardware after all.) To get to a menu item, you use the ALT key. The slow way is to press ALT-SPACEBAR and then use the right and left arrow keys to proceed to the desired top-level item. The up and down arrow keys allow you to select the desired pop-up item, if any.

You can use an accelerator keystroke of the form ALT-*x*-*y*, where *x* is the underlined letter you see in a top-level menu item and *y* is the underlined letter in the selection you want from the pop-up menu. For example, in MS-DOS, pressing ALT-F-B will display the "about" box. The programmer can choose the accelerator sequence by placing an ampersand (&) before the target letter in the resource file. Thus, in Listing 1, you see that ALT-F-P will cause the display to pause.

Using the ALT-key technique is not the only way to choose a menu item; you can also simplify the action to a single keystroke. Below the menu in Listing 1 is an accelerator table. Associated with certain keys are manifest constants also used in the menu. For example, CTRL-S in the accelerator table and "Pause" in the menu share the identifier IDM_PAUSE.

To use the table, you must first load it, usually at initialization time, with the **LoadAccelerator** call. Then, in the message loop you add the following code to inhibit translation and dispatch of any keystroke message corresponding to an accelerator:

```
if (TranslateAccelerator(...) == 0) {
   TranslateMessage(...);
   DispatchMessage(...);
}
```

TranslateAccelerator scans an incoming keyboard message for membership in the accelerator table. If it is found, it is converted to a WM_COMMAND message whose word parameter is the shared manifest constant. It thus appears to the program as if the menu has been activated. Other keyboard messages are translated and dispatched normally.

It is not necessary to have a menu item to use accelerators. In writing terminal emulators, I have used accelerators to simplify handling the function keys F1 to F12. It is even possible to specify that SHIFT, CTRL, both, or neither are to be pressed.

Message Handling in Winvue

Most Windows programs have the following simple structure:

```
WinMain()
{
    Initialization();
    MessageLoop();
}
```

When the message loop exits, so does the program.

Associated with each window in the application is a window procedure, which is called directly, not by the program, but by Windows. It has the form of a case statement:

```
WndProc()
{
    switch(message) {
        case WM_XXX:
            DoXXX();
            break;
        case WM_YYY:
            DoYYY();
            break;
        .....
        default:
    }
}
```

Practically all of the effort in writing a Windows application is in supporting the responses to those messages that must be processed by the program.

The overall feeling you get in seeing such a structure is that a Windows program basically sleeps until it is awakened by a message. This feeling is reinforced when you look at a typical message loop:

```
While (GetMessage(...)) {
    TranslateMessage(...);
    DispatchMessage(...);
}
```

Since **GetMessage** never returns until a message is available, the program is suspended. So how can you make your program carry out a continuous process?

One approach is to set the timer to wake up the application periodically. The Digital Clock presented in my article in *PJ* 6.4 provides an example. About once a second, the Clock receives a WM_TIMER message. When the message is received, the time is updated (and perhaps displayed), and the program goes back to sleep until the next message comes along. Of course, other messages can also arrive at any time, but they are usually generated by external events, such as mouse or keyboard input, information about the position and size of the window, or requests to repaint the window.

While a timer could be used in Winvue, it would be difficult to decide on the optimum time interval to display the next line of text of the file being viewed. A better approach is one that adjusts automatically to the demands of the program while still sharing CPU time with other applications:

```
While (TRUE) {
    if (PeekMessage(...)) {
        TranslateMessage(...);
        DispatchMessage(...);
    }
    else if (hFile > 0)
        ProcessFile();
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Here, *hFile* is a handle to the file being displayed.

PeekMessage and **GetMessage** can be compared to **getch** and **kbhit**. Whereas **getch** waits for a character to be entered at the keyboard, **kbhit** just checks to see if any key has been struck and returns immediately. Similarly, **PeekMessage** returns to your program (after all other applications have had their chance to run), with or without a message. If there are no messages (return value 0), then the program can use this opportunity to do something else.

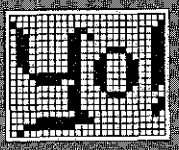
In Winvue, if a file is open, **ProcessFile** reads the next batch of characters into a buffer and then posts a WM_USER (user-defined) message to itself about where the buffer is and its length. Hence, the next time **PeekMessage** returns, there will be at least one message to process. During that time, the buffer contents are completely emptied so that the next call to ProcessFile will start with a fresh buffer. Of course, more-elaborate ways of managing the buffer can be contrived. This mechanism is adequate for Winvue and protects the contents of a data structure that is being shared between what can be regarded as two threads of execution.

But the **PeekMessage** approach has a drawback; it uses a lot of CPU cycles when there is no file to display. I can imagine a user loading Winvue as an icon to be run only occasionally. The program should be dormant unless activated by a request to display a file. By having two message functions, one to execute **GetMessage** when no file is displayed and the other executing **PeekMessage** at viewing time, we can optimize CPU sharing.

Look now at Listing 2, which shows Winvue's **WinMain** function as well as the two message loops **DoPeekMessage** and **DoGetMessage**. Loop **DoMessage** is simply a pointer to one or the other. Listing 3 shows how **ProcessFile** executes. If the user has selected the "Break" option, the file

Quality Programming Tools From
Ithaca Street Software, Inc.

FONT-TOOLS



If you develop graphics applications that include the use of text then you should definitely own Font-Tools. A complete solution to your graphics text needs, Font-Tools includes a full-screen font editor, utilities for modifying, viewing, downloading, and format converting fonts, over 60 high quality sample font files, and a high performance library with over 40 C callable functions for drawing text on the screen or into memory buffers. This product's features are too numerous to mention, so call us or write for more details. We think you'll be pleasantly surprised.

Single User $99.95 - Developer's Ed. $149.95

BABY DRIVER

Baby Driver's superb mix of features, versatility, and performance has it slated to become an industry standard for graphics printer support. It currently supports more than 150 dot matrix, inkjet, and laser printers, and additional printers are being added on a continuing basis. Its linkable object libraries allow for seamless integration with your graphics application while adding a mere 12K to its size. We've just added function support for PCL and support for true color printing is scheduled for release this summer. All things considered, a must for the serious developer.

Single User $99.95 - Developer's Ed. $199.95

ICON-TOOLS/Plus

If you think of icons as being only a specialized kind of font then you've never seen Icon-Tools. Its icon editor will remind you of our Font-Tools editor, but the similarities end there. Icons can be used to build graphics menus, and the Icon-Tools function library supports this use. Since it is customary to use a mouse in this setting, we've added mouse support. We've also added functions to produce sound and animation effects. The sample icon files provide more than 300 icons for your use. Try it and see if you agree with us that this is a nifty product.

Single User $99.95 - Developer's Ed. $125.00

1145 Ithaca Drive, Boulder, CO 80303
phone (303) 494-8865 fax (303) 494-7802

Number **226** on the Reader Service Card

The Programmer's Productivity Pack

THE MOST USEFUL PROGRAMMER'S UTILITIES TOGETHER IN ONE INTEGRATED PACKAGE!

The Programmer's Calculator
• Decimal, Binary, Octal and Hexadecimal simultaneously • Signed and Unsigned modes • SHL/SHR, SAL/SAR, ROL/ROR, RCL/RCR • AND, NOT, OR, XOR, XCHG • Memory Slots • and more!

The Keystroke Reference Center
• BIOS Scan Codes and dBASE® INKEY() values for all keys and key combinations • Portability Notes for keyboard differences • Live display of the BIOS Keyboard Status Bytes • and more!

The ASCII and Color Attribute Chart
• Displays and describes color and monochrome attributes • Enlarged display of box drawing and graphics characters • Entire ASCII and IBM extended character set!

Keystroke Recording and Playback
• Up to 100 recordings of 1,000 keystrokes each • Save recordings to disk files • Grab text from screen for use as a Keystroke Recording • and more!

plus! Screen Capture, Screen Blanking, Background Printing, Hard Disk Parking!
plus!! Memory-resident & Multitasking Versions!
plus!!! 26"x20" ASCII Poster (posters also available separately)

ONLY $79.95
plus Shipping and Handling
30 Day Money Back Guarantee

5322 Rockwood Ct. El Paso, Texas 79932

EASY TO LEARN, EASY TO USE
(915) 584-7670
Free Brochure Available

FALK DATA SYSTEMS

Number **267** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

is closed; otherwise, if the window is not an icon or the display has not been paused, the next buffer is read and posted. If end-of-file has been reached, the file is closed.

Switching between message functions takes place in the file-open and file-close routines. Again, see Listing 3. In response to a WM_COMMAND message to view a file, **WndCommand** is called with parameter IDM_-FILENAME. A dialog box is created, allowing the user to select a file. If the file exists, then **OpenTheFile** is called with the path name. If the Windows **OpenFile** function succeeds, then the name is added to Winvue's window title and **DoMessage** is set to **DoPeekMessage**. If the display is canceled or the end of the file is reached, **CloseTheFile** is executed, and the **DoMessage** is set back to **DoGetMessage**.

During display, you can change the state of the program by using the menus or the associated accelerator keys. The effects of the changes are seen on the next call to **ProcessFile**. This sort of delay between request and response is typical in Windows programs.

Using Files in Windows

OpenTheFile uses a Windows API function called **OpenFile**. Somewhat related to the low-level open function of C, **OpenFile** provides a number of unique services. One of these is to convert the ANSI (Windows) name of a file to the OEM character set, since the name is stored on disk using the latter. This can be important when using a non-English version of Windows. Other features include simple dialog box support, date and time verification, and reopen capability.

Because Windows is multitasking, problems can arise if two applications try to use the same file. In fact, it's best to have a Windows application close any open files prior to relinquishing the CPU (that is, before returning from the window procedure) and reopen them as needed. Nevertheless, in Winvue, the file is left open until its display is completed. Although I would prefer to follow the recommended approach, I have not been able to get the reopen mechanism to work as advertised. I have tried the suggestions of the *Programmer's Learning Guide* (p. 191), but I always find the position of the file to be back at the beginning. When I find an answer to this problem, I will let you know. It will probably be necessary to use some judicious combination of **reopen** and **lseek**. In the meantime, while you are waiting for an answer, don't let Winvue and another program share the same file.

Here is a scenario that should be avoided:

During Winvue's display of a file, the user selects "Pause." Then he or she opens Notepad, makes some changes, saves them, and resumes the Winvue display. When the file is finally closed by Winvue, the resulting disk file will most likely not be what was expected.

Another disaster can occur if the Windows program has a floppy disk file open for writing and the user changes the disk from another application. You can imagine what can happen to the disk.

The File Open Dialog Box Function

I'll try to make this section fairly painless. I'll begin by describing two important functions: **DlgDirList** and **DlgDirSelect**.

Introducing C_talk / Views™

Reduce C Programming for MS Windows by 75%

Introducing C_talk/Views™, a major productivity tool for Microsoft® Windows program development in C. Slash your MS Windows program development time by up to 75% through the powerfull object-oriented programming (OOP) technology of C_talk. With C_talk/Views, the most common elements of all MS Windows programs are packaged into reusable software components-object classes. Over forty software components are included as a complete applications development framework. These object classes provide an easy-to-learn and powerfully consistent interface to the rich functionality of MS Windows. C_talk/Views is also a real OOP environment. Use the software components as they are, or extend them to meet the most demanding applications.

Coming soon...C_talk/Views for the Apple Macintosh®... now your MS Windows program will port to the Mac with little or no change!

$450.00 In the US, add $5 for shipping. ($25 for Int'l handling)

CNS®

CNS, Inc.
Software Products
7090 Shady Oak Rd.
Minneapolis, MN 55344
612-944-0170, Fax 612-944-0923

. . . providing and advancing object-oriented methodology.

Number **284** on the Reader Service Card

Parallel Programming for "C"

INTERWORK™
A Concurrent Programming Toolkit

Interwork is a "C" program library which allows you to write your programs as a set of cooperating concurrent tasks. Very useful for simulation, real-time applications, and experimentation with parallel programming.

FEATURES

- Supports a very large number of tasks (typically more than 100) limited only by available memory. Low overhead per task results in very fast context switching.
- Provides a full set of inter-task communication (ITC) facilities, including shared memory, locks, semaphores, blocking queues, and UNIX™-style signals. Also has building blocks for constructing your own ITC facilities.
- Handles interrupts (DOS version) and integrates them into task scheduling. Supply your own interrupt handlers or block tasks on interrupts.
- Lets you trace task switches and inter-task communication.
- Comes with complete documentation including a user's manual and reference manual of commands.

Interwork is available for the following systems:

Hardware	*Operating System*	*Price*
IBM PC, XT, AT	PC-DOS 2.0 or later	$129
IBM PC AT	XENIX™	$159
DEC VAX™, SUN	UNIX 4.2BSD	$249

PC-DOS version is compatible with DeSmet, Lattice, and Microsoft C compilers.

Please specify hardware and operating system when ordering. Shipping and handling included; COD orders add $2.50. Send check or money order to:

Block Island Technologies
Innovative Computer Software
13563 NW Cornell Road, Suite 230, Portland, Oregon 97229-5892
(503) 241-8971

Trademarks: Interwork, Block Island Technologies; UNIX, AT&T Bell Laboratories, Inc.; XENIX, Microsoft, Inc.; VAX, Digital Equipment Corporation

Number **244** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DlgDirList takes as arguments a dialog box handle, a path specification, a list-box identifier, a static path-identifier control identifier, and a *filetype*. When called, the function fills the specified list box with a list of files matching the path specification; if necessary, a change to the new path occurs. The current path is displayed in the static path-identifier control. The *filetype* is a word parameter whose bits are used to select the class of files displayed. They can range from ordinary files to directories and drives. The function returns nonzero if a listing was made; a zero value means that the input string did not contain a valid search string.

DlgDirSelect requires a handle to a dialog box, a character buffer, and a list-box identifier. It copies the current selection into the buffer and returns nonzero if the current selection is a directory or drive name; otherwise, it returns zero.

In Listing 4, I have attempted to describe the action of the dialog box function in pseudocode. The principal tools are the two functions sketched above. The listing should facilitate your understanding of the process.

Although apparent only when reading the actual code, many considerations have also been made for correct parsing of strings in languages other than English. For those that need characters in the 128–255 range, the **AnsiUpper** function is used to convert a filename to uppercase. The code also correctly parses kanji, since movement through a string is accomplished with **AnsiPrev** and **AnsiNext** rather than by just incrementing or decrementing a character pointer.

Miscellaneous Notes

When Winvue is first loaded, it looks for a filename on the command line. The Windows command line is not broken into fields (as is *argv* in a standard C program); one is simply provided with a long pointer to an ASCIIZ string. In Winvue, this string is scanned until a null is found or the first space is reached. This piece is then passed to **OpenTheFile**.

I mentioned that the display can be started and stopped with the mouse. Here is how it is done. In the window procedure we respond to the WM_LBUTTONDOWN message by sending ourselves the appropriate WM_COMMAND message:

```
/*stop and start file display with the mouse button*/
case WM_LBUTTONDOWN:
    if (Pause)
        SendMessage(hWnd, WM_COMMAND, IDM_RESUME,0L);
    else
        SendMessage(hWnd, WM_COMMAND, IDM_PAUSE,0L);
    break;
```

This sort of thing is done frequently in Windows so that the same code does not have to be written in several places. However, you should realize that you are already in your Windows procedure and that calling it again (which is what **SendMessage** does) can cause problems if both calls are manipulating the same global or static data.

When Winvue is displaying a file, the "Pause," "Resume," and "Cancel" options on the menu are active, and the File-open and Options menus are disabled. Similarly, while waiting for the next filename, "Pause," "Resume," and "Cancel" are inactive. A single function call takes care of switching the menu attributes. By dynamically modifying the interface, you can prevent the user from attempting commands that are illegal for the current state of your program.

C Language Training Seminars

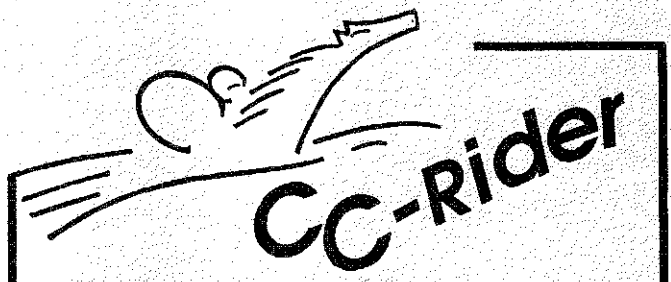
Go to C!

Embark on a voyage through C language fundamentals!

4 days of hands-on, interactive coding sessions and lectures.
Two textbooks and over 25 source code programs provided.
Focus on Microsoft C and Turbo C.
Classes held monthly at **The C-Team**.
Corporate training on location.
Tuition: $995.
Call for information.

The C-Team, Inc.
Providing C expertise for all training and programming needs.
60 East 12th St.
New York, NY 10003
212/477-3990

Number **255** on the Reader Service Card



POP UP YOUR SYMBOLS

Get *Jansi* by adding ***hypertext*** capabilities to your C editing environment. Gain access to all your macro, function, structure and typedef definitions instantly!

See CC-RIDER
SEE WHAT YOU CAN DO!

INCLUDES:

CCSYM Standard C "symbol compiler" which builds the symbol database used by CC-RIDER. Supports Microsoft C v5.1 extensions.

CCRIDER Pop-up program to recall your symbol definitions in a special window from any program.

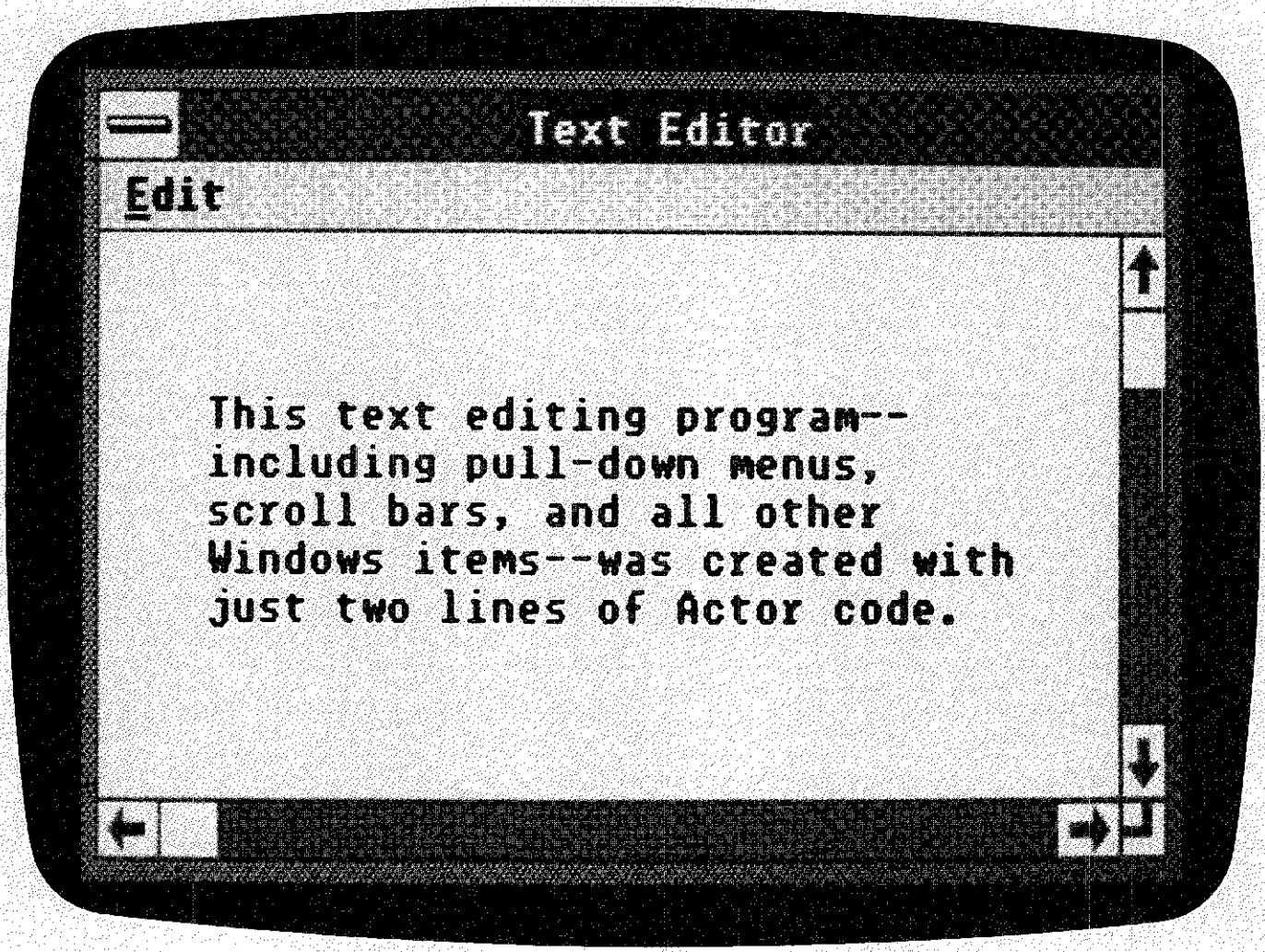
USE YOUR OWN EDITOR

Western Wares
Box C Norwood, CO 81423
(303) 327-4898 *Demo $5*

C.O.D. VISA MASTER CHARGE and AMEX
ONLY $89

Number **100** on the Reader Service Card

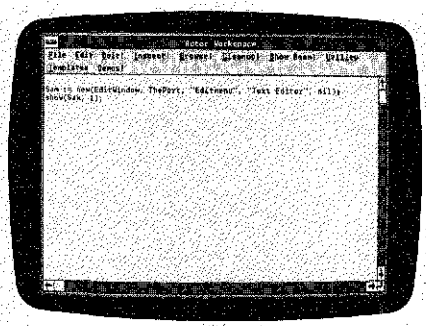
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Actor™ is the fastest, easiest way to develop applications for Microsoft® Windows. The reason—Actor's reusable toolkit of objects such as dialog boxes and edit windows. It more than doubles your overall productivity, making Actor an essential part of any Windows development project.

A full-featured, interactive, Windows-based programming environment, Actor provides immediate compilation, interactive testing, and source-code debugging. You can use it to produce fast standalone applications that support all Windows features, including Dynamic Data Exchange (DDE) and expanded memory.

Actor is a powerful, pure object-oriented programming language. It's all you need to develop complete Windows applications. You also have the option to dynamically link Microsoft C code to your Actor program.



These two lines of Actor code are all it took to produce the Windows text editing program you see above. Just think what you can do with a few more lines.

Either way, it's the fastest way to produce everything from prototypes to complete development projects. No wonder so many developers are already using Actor.

Call us now for more information.

The sooner you do, the sooner you can speed up *your* Windows development work.

Find out about Actor.
The Whitewater Group, (312) 491-2370

Technology Innovation Center
906 University Place, Evanston, IL 60201

© Copyright 1989, The Whitewater Group

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Winvue displays characters using a technique that simulates the action of a teletypewriter. I have not discussed how this is done here, but you can see my article in *PJ* 7.2 for details. You may be surprised to note that this code can also be used in PM.

Conclusion

By using a combination of **PeekMessage** and **GetMessage**, it is possible to display the contents of a file while allowing other Windows tasks to execute. Many other Windows programs also use **PeekMessage** to carry out a continuous process. Perhaps one of the best known is Cube, which draws a cube rotating about an axis. The Windows Terminal program also employs **PeekMessage**, and in *PJ* 7.1 I described a similar approach for the general problem of terminal emulation in Windows.

PeekMessage is also available in PM, and Winvue can be ported directly to this environment with little effort. However, it makes more sense to redesign the program to use a thread to carry out the action of reading a buffer of text and posting it to the window procedure. In a future article, I shall present a PM version in which such a technique is used. PJ

As a senior engineer at Olivetti Advanced Technology Center in Cupertino, California, William S. Hall specializes in Windows system support software.

Code follows.

REFERENCES

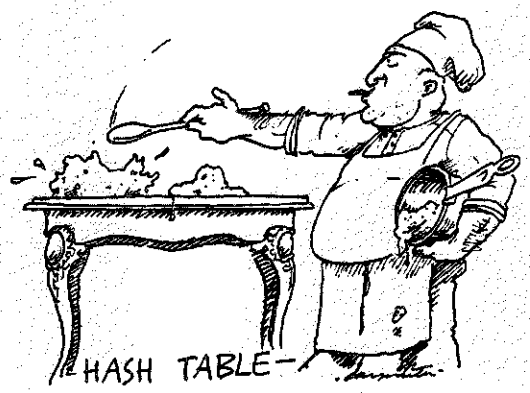
Hall, William S., "A Digital Clock for Microsoft Windows" (with M. J. Babcock), *Programmer's Journal* 6.4 (1988), 40–59.

——, "An OS/2 Presentation Manager Application Template," *Programmer's Journal* 6.5 (1988), 44–56.

——, "Smart Terminal Emulation in Windows," *Programmer's Journal* 7.1 (1989), 74–83.

——, "WPMAUX, Simple Debugging Windows for Windows and PM," *Programmer's Journal* 7.2 (1989), 56–73.

Petzold, Charles, *Programming Windows*, Microsoft Press (1988).



TurboC, Lattice C, and QuickC

"c_wndw" In Windows, Menus & File Access

Now there is an affordable, complete set of functions to produce end-user DOS applications in the C language. "c_wndw" is a very complete toolkit, providing the following facilities

- o Color and cursor control
- o Pull-down and pop-up menus, and dialog lines
- o Field formatting and string handling
- o Entry editing of numbers and dates
- o Stacked, overlapping windows
- o File access error trapping and handling
- o Access to dBASE III plus '.DBF' files
- o Hashed random file access
- o Function key mapping and control
- o Undo and Help capability

For a license fee of $35, we provide a full set of object libraries for a C compiler, the full manual, and 19 sample programs. (S&H $2, GA res. add $1.85 tax). Specify your C compiler. Source code is available for $95.

To find out more about the "c_wndw" toolkit, write or call for information to Marietta Systems - (404) 565-1560
P.O. Box 71506, Marietta, GA 30007 Visa-M/C

Trademarks: "c_wndw", dBASE, QuickC, Turbo C, Lattice C

Number 223 on the Reader Service Card

MULTI-USER

Get dBASE IV Source...

Code Base 4

The "C" Library for Database and Screen Management

WINDOWS
- Data Entry
- Menus

EXTENSIVE
- dBASE IV functionality
- Unlimited file opens

EFFICIENT
- Fast B+ tree indexing
- Small EXE files

COMPATIBLE
- Use dBASE or Clipper data and index files

Add the Speed and Flexibility of "C" to the Productivity of dBASE with a 30-day money-back guarantee. Only $295.

For a FREE DEMO Call Today! (403) 439-8171

SEQUITER SOFTWARE INC.

Number 269 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from ". . . TYPE Command . . ." by William S. Hall

1

LISTING 1

```
/* Excerpts from Winvue.rc */

/* menu */
Winvue MENU
{
    POPUP "&Files" {
        MENUITEM "&View...\tF1", IDM_FILENAME
        MENUITEM "&Pause\t^S", IDM_PAUSE,GRAYED
        MENUITEM "&Resume\t^Q", IDM_RESUME,CHECKED,GRAYED
        MENUITEM "&Cancel\t^C", IDM_CANCEL,GRAYED
        MENUITEM Separator
        MENUITEM "&About...", IDM_ABOUT
    }
    POPUP "&Fonts" {
        MENUITEM "&System", IDM_SYSFONT, CHECKED
        MENUITEM "&Terminal", IDM_OEMFONT
    }
    POPUP "&Options" {
        MENUITEM "&Mask 8th bit", IDM_MASK
    }
}

/* menu accelerators */
Winvue ACCELERATORS
{
    VK_F1, IDM_FILENAME, VIRTKEY
    VK_CANCEL, IDM_CANCEL, VIRTKEY, CONTROL
    "^C", IDM_CANCEL
    "^S", IDM_PAUSE
    "^Q", IDM_RESUME
}

/* File open dialog box */
DT_OPEN DIALOG  20,20,148,80
STYLE WS_POPUP | WS_DLGFRAME
{
    EDITT       EXTIDD_EDIT, 46,2,100,12,WS_TABSTOP | WS_GROUP | ES_AUTOHSCROLL
    LISTBOX     IDD_LISTBOX, 4,20,70,56,LBS_STANDARD | WS_TABSTOP | WS_GROUP
    DEFPUSHBUTTON  "View",      IDOK,       87,40,50,14,WS_GROUP
    PUSHBUTTON     "Cancel"     IDCANCEL,   87,58,50,14,WS_GROUP
    LTEXT          "",          IDD_PATH,   78,20,60,12
    RTEXT          "Filename:"  -1,         2,2,40,12
}
```

LISTING 2

```
/* excerpt from WINVUE.C */

/* entry point for windows */
int FAR PASCAL WinMain(hInstance, hPrevInstance, lpszCmdLine, cmdShow)
HANDLE hInstance, hPrevInstance;
LPSTR lpszCmdLine;
int cmdShow;
{

  /* initialize program */
    if (!InitProgram(hInstance,hPrevInstance, lpszCmdLine, cmdShow))
      /* if we fail, then exit */
        return FALSE;

  /* fall into the message loop */
    while (TRUE)
        DoMessage();

}
```

2

```
/* execute GetMessage while waiting for a file to view */
void NEAR DoGetMessage()
{
    MSG msg;

    if (GetMessage((LPMSG)&msg,NULL,0,0)) {
        if (TranslateAccelerator(MWnd.hWnd,hAccel,(LPMSG)&msg) == 0) {
            TranslateMessage((LPMSG)&msg);
            DispatchMessage((LPMSG)&msg);
        }
    }
    else
        exit((int)msg.wParam);
}

/*
   Execute PeekMessage while displaying a file so that other Windows
   programs can continue to run.
*/
void NEAR DoPeekMessage()
{
    MSG msg;

  /* if there is a message, then process it */
    if (PeekMessage((LPMSG)&msg,NULL,0,0,PM_REMOVE)) {
        if (msg.message == WM_QUIT)
            exit((int)msg.wParam);
        if (TranslateAccelerator(MWnd.hWnd,hAccel,(LPMSG)&msg) == 0) {
            TranslateMessage((LPMSG)&msg);
            DispatchMessage((LPMSG)&msg);
        }
    }
  /* otherwise if file is still open process it */
    else if (hFile > 0)
        ProcessFile();
}
```

LISTING 3

```
/* extract from WINVUEFN.C */

/* read next buffer of data from file or close the file */
void NEAR ProcessFile()
{
    int numread, size;
    size = min(MWnd.MaxCols, MAX_STR_LEN);

  /* user canceled the file display */
    if (Break) {
        hFile = CloseTheFile(TRUE);
        Break = FALSE;
    }
    else if (!(Pause || IsIconic(MWnd.hWnd))) {
      /* read next buffer of data */
        if ((numread = read(hFile, instr, size)) == 0)
      /* no more data (or perhaps a read error) */
            hFile = CloseTheFile(FALSE);
        else
      /* post the buffer and its length for later processing */
            PostMessage(MWnd.hWnd, WM_USER, numread, (LONG)(LPSTR)instr);
    }
}


/* process the menu */
void NEAR WndCommand(HWND hWnd, WORD wParam)
{

    HMENU hMenu = GetMenu(hWnd);
    HANDLE hInstance = GetWindowWord(hWnd, GWW_HINSTANCE);
    int result;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    switch (wParam) {

        case IDM_FILENAME:
            result = DialogBox(hInstance, MAKEINTRESOURCE(DT_OPEN),hWnd,
                        MakeProcInstance((FARPROC)DlgFnOpen, hInstance));
          /* file has been found */
            if (result) {
                if ((hFile = OpenTheFile(szFileNameSave)) != -1) {
                    TTYClear(&MWnd);
                    EnableMenus(hMenu, FALSE);
                    Pause = FALSE;
                }
            }
            break;

      /* suspend file display */
        case IDM_PAUSE:
            Pause = TRUE;
            CheckMenuItem(hMenu, IDM_PAUSE, MF_CHECKED);
            CheckMenuItem(hMenu, IDM_RESUME, MF_UNCHECKED);
            break;

      /* resume file display */
        case IDM_RESUME:
            Pause = FALSE;
            CheckMenuItem(hMenu, IDM_RESUME, MF_CHECKED);
            CheckMenuItem(hMenu, IDM_PAUSE, MF_UNCHECKED);
            break;

      /* cancel file display */
        case IDM_CANCEL:
            Break = TRUE;
            break;

        ...

    }
}

/* open the file, display its name, and change the message function pointer */
int FAR OpenTheFile(char *pfilename)
{

    int hfile;
    char winname[30];

  /* open the file */
    hfile = OpenFile(pfilename, (LPOFSTRUCT)&fstruct, OF_READ);

    if (hfile != -1) {
      /* change the window title */
        strcpy(winname, szWinTitle);
        strcat(winname, strrchr(fstruct.szPathName,'\\') + 1);
        SetWindowText(MWnd.hWnd, winname);
      /* change message function pointer */
        DoMessage = DoPeekMessage;
    }
    else
      /* open failed */
        ShowMessage(MWnd.hWnd,pfilename,IDS_CANNOT_OPEN,MB_OK | MB_ICONEXCLAMATION);
    return(hfile);
}

/* close the file, enable inactive menus, and switch message processor */
static int NEAR CloseTheFile(param)
{

    HMENU hmenu = GetMenu(MWnd.hWnd);

    Pause = FALSE;              /* reset the pause button */
    EnableMenus(hmenu, TRUE);   /* enable grayed menus */

    DoMessage = DoGetMessage;   /* change the message processing pointer */

  /* show the result to the user */
```



```
    if (param)
        ShowMessage(MWnd.hWnd,(char *)NULL,IDS_CANCEL, MB_OK | MB_ICONEXCLAMATION);
    else
        ShowMessage(MWnd.hWnd,(char *)NULL,IDS_DONE, MB_OK | MB_ICONASTERISK);
    return(close(hFile));
}
```

LISTING 4

```
/* Pseudo code for the file open dialog box manager */

DlgFileOpen(HWND hWnd, unsigned msg, WORD wParam, LONG lParam)
{

    switch(msg) {
        case WM_INITDIALOG: /* sent by windows before the box is drawn */
            Set edit text field to the default search specification (= *.txt).
            Limit the amount of text which can be typed in the edit box
                to the length of a path name minus the length of an extension.
            Use DlgDirList() to fill the list box with file names which
                match the current search specification.

        case WM_COMMAND:  /* the user selected a control */
            switch(wParam) {
                case IDOK:  /* user selected the OK button or hit RETURN */
                    Get the contents of the edit box.
                    Add the default extension to the contents if possible.
                    if the modified contents contain a search spec, use
                        DlgDirList() to try to display a matching file list
                        in a possibly new directory.  If successful, break.
                    if no matching file list but contents contain
                        a search spec, then sound warning and break.
        Loadit:
                    Convert contents to upper case and check for legal
                        file name.  If OK, end dialog box and
                        return TRUE.  Otherwise show error message and break.

                case IDCANCEL: /* user pressed CANCEL or hit ESC */
                    End dialog box and return FALSE.

                case ID_LISTBOX: /* user selected a line from the list box */
                    switch(HIWORD(lParam)) {
                        case 1:         /* single mouse click */
                            Get the contents of the edit box.
                            Use DlgDirSelect() to retrieve the selected item
                                from the list box.
                            If the selection is a file name,
                                display it in the edit box and break.
                            If the list box selection is a disk drive
                                or directory, append the edit box contents
                                if the latter contains any wild cards.
                                Otherwise just append the default extension.
                                Display modified selection in the edit box
                                and break.
                         case 2:/* mouse double click */
                            Get the contents of the edit box.
                            Use DlgDirList() to try to find a matching search
                                path.  If successful, break.
                                If not go to Loadit

                    }

                case ID_EDIT: /* user typed in the edit box */
                            if no characters in edit box, disable OK button
                                else enable it.

            }
    }
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

User Interface Tools for DOS

by
Vaughn Vernon

In earlier articles, I concentrated on using C++ under OS/2. However, it is obvious that more development is under way for MS-DOS applications. So, I want to shift gears a little and show you how to take advantage of C++ and OOP under MS-DOS. The classes offered here may be quickly and easily ported to OS/2.

User Interface Classes

One of the most important steps in developing any application is designing the human interface—the part of the application the user sees and through which the user tells the application what to do. For this article, I developed three major classes that help a C++ programmer interface with the user: the Display class, the Mouse class, and the Keyboard class. Along with these major classes, I also developed some building-block and helper classes that are used with the major classes.

The Display Class The objects of class Display are used to control the output of data to the PC's screen. When you think of creating the Display class, you probably say, "Hey, I'll just throw in some protected variables to represent the cursor row and column, as well as some variables that define the size of the screen (for example, 25x80)." But if we took this approach, we would not be taking advantage of several C++ strengths. Remember, when you use C++, you have the power of inheritance in your hands, as well as the ability to use building blocks to simplify your job. By using building blocks, you will tend to reuse code rather than reinvent the wheel.

Now that you have the idea, we will create a Cursor class and a ScreenSize class. You might well do that—but what would you be able to use a Cursor class and a ScreenSize class for other than cursors and screen sizes? If we thought a little more abstractly, we could come up with classes like Point and Rectangle. And I can think of at least ten uses for each of these classes in addition to defining a cursor and a screen size—in fact, the Rectangle class itself uses objects of class Point to define its area.

Listing 1 contains the class descriptions for class Point and class Rectangle. A Point has an *x* coordinate and a *y* coordinate and a few methods for setting, retrieving, testing, and otherwise manipulating the coordinates. For instance, by using overloaded operators, you can assign (=) and compare (==) Points:

```
Point p1(35,99);
Point p2 = p1;

if (p1 == p2)
       // ...
```

I chose not to implement some other operators because of space limitations, but you might want to supply the addition (+, +=) and other operator methods on your own.

The Rectangle class uses two Points, an *origin point (op)* and a *corner point (cp)*, to define its location and size. A Rectangle's origin is its upper left coordinate, and its corner is its lower right coordinate. Rectangle methods allow you to reset and query a Rectangle's origin and corner points. You can also find out the *height* and *width* of a Rectangle. Additionally, the *extent* of a Rectangle is a Point that defines its height and width as *x* and *y* coordinates, respectively.

A curious method in both Point and Rectangle class is **draw**. This routine is meant only as protocol; in other words, it is meant to stipulate that a Point and a Rectangle cannot themselves be drawn but that a class derived from them can be. Therefore, classes derived from Point and Rectangle should supply **draw** methods of their own when applicable.

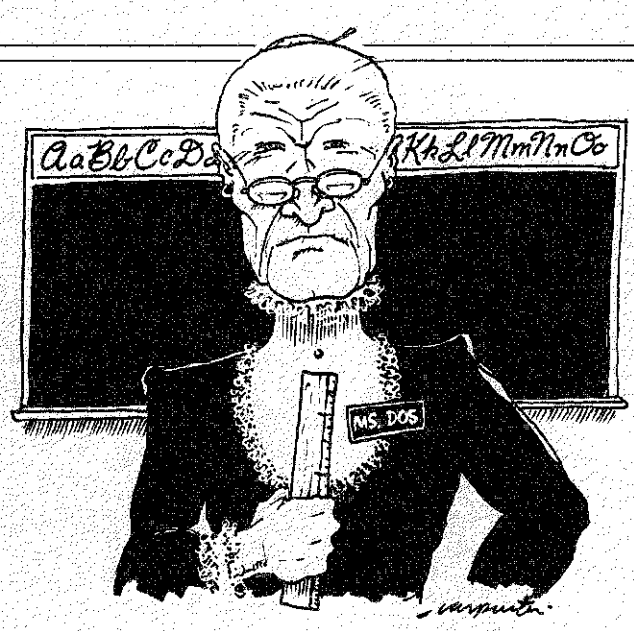
The Display class definition and methods are provided in Listing 2. Display is derived from Rectangle, and it contains an object of class Point that defines the cursor. The Display class has some other protected objects, which will be mention later.

I probably should have created a Cursor class derived from class Point. The Cursor class would have known how to **draw** itself and turn itself off and on (hide and display). In that case, the Display's cursor would be of class Cursor instead of class Point. Again, space prevented me from being too elaborate, but I told on myself for your benefit.

In this implementation, you set the display cursor by using the () operator. For instance, **disp(0,0);** sends the cursor home. You can also send a Point, rather than row and column values, as the argument.

The Display class needs to be able to protect itself from "mouse tracks" when displaying data. Mouse tracks occur if the mouse pointer is not hidden while the screen is being updated (see your mouse technical reference). For that reason, a Display object needs a pointer to a Mouse object so that it can tell the mouse to hide itself during a screen refresh and to show itself again after the refresh is completed. The mouse pointer is supplied when a Display object is constructed.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Display objects also have a default video attribute. Video attributes and colors are supplied via the DisplayAttr class. To use a particular attribute, create a DisplayAttr object, and use the **set** method:

```
Mouse mouse;
Display disp(&mouse);
DisplayAttr attr;

attr.set(attr.reverse());
disp.defaultAttr(attr);
```

The other methods in the DisplayAttr class define the various monochrome attributes and CGA/EGA/VGA-compatible colors. To find out if the current display mode is color or monochrome, send the **hasColor** message to the Display object and check for a true or false answer.

You can send strings or characters to the Display object for display in one of two ways: by sending the **putString** or **putChar** message, or by using the C++ streams-style method, <<:

```
disp << "hello" << ',' << 'w' << "orld!";
```

Notice the definition for the overloaded << operators. They both return a reference to the message-receiving object, allowing the recursive use of the same Display in one set of statements, as shown above. In other words, since

```
disp << "hello";
```

returns a reference to "disp," the next

```
... << ',';
```

message has an object to send to.

Finally, the **draw** method makes sure the screen is fully up to date. If you peek into the **putBlock** and **putChar** methods, you will see that this version automatically does a fast write to the physical display. But having a **draw** method is good for operating systems that provide slow screen updates or buffering, such as UNIX. In the case of UNIX, all physical screen output would be delayed until the **draw** message was received.

The Mouse Class Mice are becoming increasingly popular in both graphics and text environments. My Mouse class allows you to use mice under a text environment. Listing 3 contains the Mouse class description and methods.

When you declare an object of class Mouse, the constructor takes care of automatically displaying the mouse pointer on the screen. The **buttons** method answers the number of buttons on the physical mouse device (2 or 3).

The **hide** and **show** methods manage whether or not the mouse pointer can be seen at all. As mentioned earlier, Display objects must protect the screen from mouse tracks by sending these two messages to the mouse at the appropriate time.

You use the **getEvent** method to retrieve a mouse event, such as a click or a drag. You supply arguments to **getEvent** for retrieving event information. The Point argument receives the mouse pointer location information. The remaining three arguments are Boolean-type arguments that get set according to the status of the 3 (or 2) mouse buttons. If a button is pressed, its corresponding flag is set to true; otherwise, it is set to false.

The **getEvent** method uses a filtering test to help prevent you from receiving redundant mouse events. If you send the **getEvent** message to a mouse and the event is a duplicate of the previous event, **getEvent** answers false (no event pending); otherwise, the Point and button information are filled in, and **getEvent** answers true.

The **flush** message is sent to a mouse when you wish to clear the mouse event queue of all events. The mouse constructor sends itself the **flush** message so that all mice start off with a clean queue. You send the **setPosition** message to a mouse to have its pointer placed at a particular position. The argument is a Point, of course.

The **makePointer** method, like the Point and Rectangle **draw** method, is for protocol only; in other words, a derived class of the Mouse class can implement a hardware-dependent method for changing the appearance of the mouse pointer on the display. The void pointer argument points to an application-specific or hardware-specific data structure that defines the shape or attributes the mouse should use.

The Keyboard Class My final set of classes provides tools for using the keyboard. Listing 4 contains the files keyboard.cls and keyboard.cpp, which implement the Keyboard class and the KeyTypes class.

Objects of Keyboard class are self-contained; that is, they not only know how to input keystrokes, they also know how to hold on to the current keystroke and test the current keystroke against another key.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Depending on its argument, the **input** method can wait on input or return immediately if a keystroke is not available. Pass a zero if you don't want to wait or a nonzero if you wish to wait infinitely. The **input** message answers the current keystroke or a -1 if no input is available.

By sending the **value** message to a Keyboard object, you receive the results of the latest **input** message. This saves you the trouble of creating an integer to store each keystroke.

The >> method sends the **input(1)** message to the Keyboard receiver. It also plays some special tricks. For instance,

```
int k1, k2, k3;
Keyboard kb;

kb >> k1 >> k2 >> k3;
```

places three consecutive keystrokes into *k1*; *k2*, and *k3*, respectively. Just as the Display << method returns a reference to the messaged Display object, so the Keyboard >> method returns a reference to the messaged Keyboard object.

You can flush all pending input from a Keyboard by sending it the **flush** message. Flushing pending keystrokes comes in handy when a user hits the PAGE DOWN key in rapid fire and your application is having trouble keeping up.

The KeyTypes class is used to test input keystrokes with common standard key values, such as F1:

```
Keyboard kb;
KeyTypes kt;

disp.putString(Point(0,0), "Press F1", anAttr);
disp.draw();

if (kb.input(1) != kt.F1()) {
        disp.putString(Point(disp.height(),0),
                "Error", errAttr);
disp.draw();
}
```

Here, the Keyboard object tests its current input against the F1 key. I include most of the popular key values for the PC in the KeyTypes class. However, I left out the values for function keys and keypad keys used with CTRL and ALT.

The Dots Program

Now that we have some classes to work with, I give you a simple application that uses all of them. The application is called Dots. Dots allows you to draw simple (more like primitive) pictures on the display with the mouse. When you click the left mouse button, a dot is drawn where the mouse is located. If you press the right button (or middle button, if you have a three-button mouse) and the mouse pointer is over a dot, the dot is removed. Naturally, you can drag the mouse with a button pressed to make drawing much faster. To leave Dots, press ESC.

I was amazed at the amount of utility contained in the set of classes presented here. In less than one page of C++ code (including comments), I came up with a highly functional—and almost useful—application.

Odds and Ends

I tried something that should cause some trouble this time around. It is related to the name extensions I gave the class definition files: CLS. I did this because I have been plagued by the number of different file-naming conventions introduced by the various translator/compiler writers/porters. I didn't do it to introduce yet another convention to remember; rather, I wanted to find out what the rest of the C++ community thinks about this nuisance.

For example, there are three major C++ implementations for MS-DOS: Guidelines C++, OASYS/Glockenspiel C++, and Zortech C++. Each of these uses a different file extension for the class definition files:

Guidelines:	.cpp	.h
Glockenspiel:	.cxx	.hxx
Zortech:	.cpp	.hpp

And, to beat it all, UNIX users generally just use c and h! As you can see, trying to port a large (or even small) class library between these three implementations is a nightmare, because of class definition files alone. This really confuses things, especially if you are trying to write tools that need to read C++ sources. Such a tool needs to be very language sensitive just to tell if it's reading a C++ source or a plain C source, since the filename doesn't tell you what it is. I'm not the first person to notice this or to complain about it. For instance, I recently read an article by Bjarne Stroustrup, who wants to find a way to standardize include files for C++ as well as to standardize C++ in general.

I introduced the CLS (or cls) extension to give implementors something to think about. I feel that a C++ class definition file is not really a header file—at least not as C originally knew it. It is a class definition. So what do you name such a file? Giving it a C extension (for Class) surely won't do it—C is already used. I used CLS because the file defines a class or group of classes. But, frankly, I don't know what kind of problem CLS would introduce for operating systems other than MS-DOS, OS/2, and UNIX/XENIX.

How about CD for Class Definition, or just D? I don't know! I would be willing to settle for hpp or hxx, even if those extensions are not truly descriptive. I just wish someone would say what it should be. It would surely make the job of supporting class libraries for different translators and compilers much easier. I would also imagine that developers working on intelligent tools, such as execution environments, could use some help in this area too.

Compilers/Translators I have just a few comments to make about the Zortech compiler—after all, I *did* promise to talk more about it when I knew more about it.

I have mixed feelings about the Zortech compiler. It is much faster, of course, than a translate-then-compile setup. It can handle much larger files than the AT&T translator, because the compiler itself uses less memory to load than the AT&T translator does. And because it uses less memory, you can use Make to build your application. These are three big reasons why I would consider recommending Zortech.

However, there are other considerations involved. Because Zortech is a compiler and not a ported AT&T translator, it is bound to have a few of its own quirks. If portability is a factor for you, I know that you will have problems porting code from a translator to Zortech or vice versa. One example is that overloaded in-line functions (not class methods) do not work. Period. And the compiler itself is a little shaky: I recently caused two compiler bugs (where the compiler dies with an "unknown" problem) that would have been accepted or at least flagged as errors by a translator. I know of several other outstanding problems.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

But not all is well with the Guidelines C++ port, either. For one thing, the translator is a memory hog. (So is the Glockenspiel translator.) You cannot use Make to build your application. You must use predefined batch files or the translator will run out of memory. (Next time I will explain my solution to this problem.) The number of in-line functions is limited due to the amount of memory required to translate them. Also, you cannot use explicit far pointers with Guidelines, while you can with both Zortech and Glockenspiel.

For the code in this article I used the Guidelines translator and then modified the code to work with Zortech. I wish I could have done it the other way; it would have saved me a lot of time. PJ

Vaughn Vernon is president of Aspen Scientific, producers of CURSES and FORMATION, user interface cross-development tools for UNIX, MS-DOS, and OS/2. Vaughn is co-author of The Advanced C Programmer's Guide to OS/2, *published by Microsoft Press.*

Code follows.

VENDORS

Guidelines Software
P.O. Box 749, Dept. PJR
Orinda, CA 94563
415-254-9183 or 9393

OASYS
230 Second Ave.
Waltham, MA 02154
617-890-7889

Zortech, Inc.
366 Massachusetts Ave.
Arlington, MA 02174
617-646-6703

What Is C++?

(A Videotape Presentation)

Considering C++ for your next project? Confused after trying to read a C++ book? Or just curious about all the hOOPla? This VHS-format videotape, featuring author/speaker William M. Miller, gives a comprehensive and understandable overview of the major features of C++ and Object-Oriented Programming.

- On-line access to lecturer (via BIX)
- Moneyback guarantee!

Send $19.95 (+$4.00 S&H) for 1-week rental or $59.95 for purchase (postpaid) to:

Software Development Technologies, Inc.
Dept. 401
P. O. Box 366
Sudbury, MA 01776

Or call (508) 443-5779, 8:00 a.m. to 6:00 p.m. Eastern time, for Visa/Mastercard orders.

Inquire about our full C++ training course, available for on-site or videotape delivery.

Number **275** on the Reader Service Card

4c™ A *ZOOM* key for your editor!

Hypertext editing for Programmers

Explore source code. Navigate uncharted logic trails easily!

4 c includes:

Now With **Pascal** Support!

★ **A Source Code Analyzer:**
The analyzer maps and cross-references functions and variables to their respective path and file names. The cross-reference table can be displayed alphabetically and by type of variable. It allows a programmer to edit functions directly without specifying a file name.

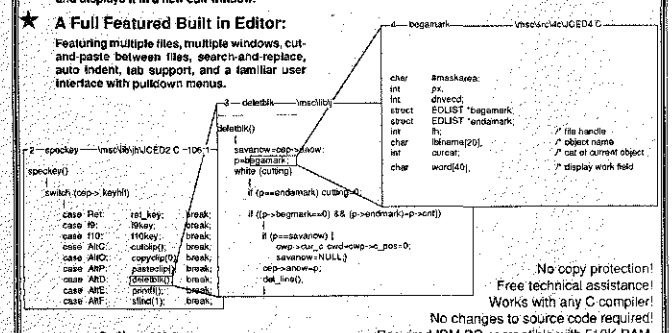
★ **A Point-and-Shoot Zoom Key:**
Which instantly locates any function, global variable, structure definition, #define statement, or macro, etc., in any file, directory, and drive, and displays it in a new edit window.

★ **A Full Featured Built in Editor:**
Featuring multiple files, multiple windows, cut-and-paste between files, search-and-replace, auto indent, tab support, and a familiar user interface with pulldown menus.

"Given a procedure's identifier, an editor should be able to bring up the full procedure interface, including parameters, so that you don't have to remember how many and in what order. Wouldn't it be nice to hit a hot key on the name of a complicated data type and have its definition pop up in a window? It may seem like a real small matter, but how many of you still flip through printouts doing things the editor should be doing?"

- Philippe Kahn, "The Connection", Spring/Summer 1988

4 c does this and more!



No copy protection!
Free technical assistance!
Works with any C compiler!
No changes to source code required!
Required IBM PC compatible with 512K RAM.

Call or Write:

Tri-Technology Systems Inc.
1225 S. Elgin, Forest Park, IL 60130
312-366-7595

30 Day Money Back Guarantee
Only $119.00 plus shipping
Visa, Mastercard, check, money order, or COD.

Number **276** on the Reader Service Card

Are you still fussing with floppies for BACKUP? Our QIC40 JUMBO tape drive from MISOSYS is your solution!

The Colorado Memory Systems' JUMBO tape drive is one drive that fits all computers. It comes ready for internal use in AT's, XT's, and PC's, and connects to your floppy disk controller. Our adaptor board can be used to connect JUMBO when all FDC connectors are in use. Kits convert Jumbo to external use.

- In about 5.5 minutes, JUMBO backs up 10MB's file-by-file - the fastest in the industry! 40MB's gets backed up in about 18 to 20 minutes. Uses industry-standard DC2000 tape cartridges.
- JUMBO plugs into your floppy disk controller to save cost, power, and a slot. Needs 5-1/4" (or 3.5" with faceplate) mounting slot.
- Our tape adapter board mounts in your host computer to provide two more floppy ports — especially useful in 286 and 386 computers. When used with our external JUMBO, it lets you share your JUMBO drive between computers.
- JUMBO has custom chips, high speed brushless motors, automatic circuits, no pots to vibrate out of calibration, and fewest parts to assure long life and technical leadership for years to come.
- By using advanced Reed-Soloman error correction, JUMBO typically backs up more than 100,000 cartridges before encountering its' first error — of less than one error per 1000 years if the cartridge is fully backed up every working day.

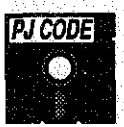
Jumbo	$295 ($5S&H)
Tape Adaptor	$75 ($3S&H)
External Adaptor	$110 ($5S&H)
DC2000 tape	$22.50 ($2S&H)

M/C & VISA. S&H are U.S. only.

MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170-0239
800-MISOSYS
or 703-450-4181

Number **238** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



All code in this issue is available on 360K, MS-DOS disks.
ORDER BY PHONE — VISA/MC orders call toll free 800/234-0386.
Mail order information on pages 8 and 88.

Code from "User Interface Tools . . ." by Vaughn Vernon

1

LISTING 1

```
// point.cls    a C++ Point class.  defines an x,y coordinate point
//
// x,y mapped: (like saying x=row, y=col)
//
// 0,0 +---------- y ->
//     |
//     |
//     x
//     |
//
// © Aspen Scientific 1989. All Rights Reserved.
// Author: Vaughn Vernon

#ifndef _POINT_CLASS

# define _POINT_CLASS   1

class Point {
        int     xc, yc;
public:
        int     x()             { return xc; }
        int     y()             { return yc; }
        int     x(int xx)       { return (xc = xx); }
        int     y(int yy)       { return (yc = yy); }

        Point & operator=(Point & p) { xc = p.xc; yc = p.yc; return *this; }
        Point & operator()(int xx, int yy) { xc = xx; yc = yy; return *this; }
        int     operator==(Point & p) { return (xc == p.xc && yc == p.yc); }
        virtual void draw()     { }

        Point(int xx =0, int yy =0)     { xc=xx; yc=yy; }
        ~Point()                { }
};

#endif

// rect.cls     Rectangle class.  uses two Points to define a rectangle.
//
// © Aspen Scientific 1989. All Rights Reserved.
// Author: Vaughn Vernon

#ifndef _RECTANGLE_CLASS
# define _RECTANGLE_CLASS       1

# include "point.cls"

class Rectangle {

protected:
        Point   op;     // origin Point
        Point   cp;     // corner Point
public:
        Rectangle(Point & o, Point & c) { op = o; cp = c; }
        Rectangle()             { op = Point(0,0); cp = Point(0,0); }
        ~Rectangle()            { }

        Point & origin()        { return op; }
        Point & origin(Point &p){ return op = p; }
        Point & corner()        { return cp; }
        Point & corner(Point &p){ return cp = p; }
        Point   extent()        {
                Point p(cp.x() - op.x(), cp.y() - op.y());
                return p;
        }
        int     height()        { return cp.x() - op.x(); }
        int     width()         { return cp.y() - op.y(); }
```

2

```
        // draw() is protocol: derived class responsibility
        virtual void draw()     { }
};

#endif
```

LISTING 2

```
// display.cls  a C++ Display class for placing data on the display.
//
// © Aspen Scientific 1989. All Rights Reserved.
// Author: Vaughn Vernon

#ifndef _DISPLAY_CLASS

# define _DISPLAY_CLASS_        1
# include "mouse.cls"
# include "point.cls"
# include "rect.cls"

// DisplayAttr class

typedef unsigned int    Attribute;

class DisplayAttr {

protected:
        Attribute       attribute;
public:
        // set like this:
        //
        // DisplayAttr attr;
        //
        // attr.set(attr.backGreen() | attr.foreBlack());

        Attribute       set(Attribute a) { return attribute = a; }
        Attribute       get()           { return attribute; }

        // generic/monochrome attributes
        Attribute       blink()         { return 0x8000; }
        Attribute       bold()          { return 0x0800; }
        Attribute       normal()        { return 0x0700; }
        Attribute       reverse()       { return 0x7000; }
        Attribute       underline()     { return 0x0100; }

        // color attributes
        Attribute       foreBlack()     { return 0x0000; }
        Attribute       foreBlue()      { return 0x0100; }
        Attribute       foreGreen()     { return 0x0200; }
        Attribute       foreCyan()      { return 0x0300; }
        Attribute       foreRed()       { return 0x0400; }
        Attribute       foreMagenta()   { return 0x0500; }
        Attribute       foreBrown()     { return 0x0600; }
        Attribute       foreWhite()     { return 0x0700; }
        Attribute       foreGray()      { return 0x0800; }
        Attribute       foreLtBlue()    { return 0x0900; }
        Attribute       foreLtGreen()   { return 0x0a00; }
        Attribute       foreLtCyan()    { return 0x0b00; }
        Attribute       foreLtRed()     { return 0x0c00; }
        Attribute       foreLtMagenta() { return 0x0d00; }
        Attribute       foreYellow()    { return 0x0e00; }
        Attribute       foreBrightWhite() { return 0x0f00; }

        Attribute       backBlack()     { return 0x0000; }
        Attribute       backBlue()      { return 0x1000; }
        Attribute       backGreen()     { return 0x2000; }
        Attribute       backCyan()      { return 0x3000; }
        Attribute       backRed()       { return 0x4000; }
        Attribute       backMagenta()   { return 0x5000; }
        Attribute       backBrown()     { return 0x6000; }
        Attribute       backWhite()     { return 0x7000; }
```

Code continues . . .

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE FOR THE PC

source code, of course

	MS-DOS File Compatibility Package (create, read, & write MS-DOS file systems on non-MS-DOS computers)	$500
	Bluestreak Plus Communications (two ports, programmer's interface, terminal emulation)	$400
	CQL Query System (SQL retrievals plus windows)	$325
	GraphiC 4.1 (high-resolution, DISSPLA-style scientific plots in color & hardcopy)	$325
	PC Curses (Aspen, Software, System V compatible, extensive documentation)	$290
NEW!	Code Base (database manager, dBase and Clipper compatibile indes & data files; Version 4.0)	$260
	Greenleaf Data Windows (windows, menus, data entry, interactive form design; specify compiler)	$220
	Greenleaf Communications Library (interrupt mode, modem control, XON-XOFF; specify compiler)	$175
	TurboTEX (TRIP certified; HP, PS, dot drivers; CM fonts; LaTEX)	$170
	Sherlock (C debugging aid)	$170
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$165
NEW!	AT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for ATs)	$160
	Greenleaf Functions (296 useful C functions, all DOS services; specify compiler)	$160
	WKS Library Version 2.0 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	OS/88 (U**x-like operating system, many tools, cross-development from MS-DOS)	$150
	ME Version 2.1 (programmer's editor with C-like macro language by Magma Software; Version 1.31 still $75)	$140
	Vmem/C (virtual memory manager; least-recently used pager; dynamic expansion of swap file)	$140
	Turbo G Graphics Library (all popular adapters, hidden line removal)	$135
	TurboGeometry (library of routines for computational geometry)	$125
Updated!	Install 2.1 (automatic installation program; user-selected partial installation; CRC checking)	$120
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows)	$105
	Minix Operating System (U**x-like operating system, includes manual)	$105
NEW!	HyperText Viewer (simple hypertext system; multi-file documents; includes Tiny Curses)	$100
	PC/IP (CMU/MIT TCP/IP implementation for PCs)	$100
	B-Tree Library & ISAM Driver (file system utilities by Softfocus)	$100
	Tele Operating System (TeleKernel, TeleWindows, TeleFile, & TeleBTree by Ken Berry)	$100
	The Profiler (program execution profile tool)	$100
	QC88 C compiler (ASM output, small model, no longs, floats or bit fields, 80+ function library)	$90
	C Windows Toolkit (pop-up, pull-down, spreadsheet, CGA/EGA/Hercules)	$80
	JATE Async Terminal Emulator (includes file transfer and menu subsystem)	$80
	Polyglot Lisp-to-C Translator (includes Lisp interpreter, Prolog, and simple calculus prover)	$80
	MultiDOS Plus (DOS-based multitasking, intertask messaging, semaphores)	$80
NEW!	evalO (C function to evaluate ASCII infix expression string; 17 built-in functions)	$75
	Kinetic Image Synthesizer (3-D animation system ... Saturday morning on your PC!)	$75
	XT BIOS Kit (roll your own BIOS with this complete set of basic input/output functions for XTs)	$75
	Professional C Windows (lean & mean window and keyboard handler)	$70
	Heap Expander (use LIM-standard expanded memory as an extension of the heap)	$65
	lp (flexible printer driver; most popular printers supported)	$65
	Icon-Tools (full-featured icon display and editing system)	$65
NEW!	SYSKIT (rommable or TSR debug/monitor; easily expandaded)	$60
	Quincy (interactive C interpreter)	$60
NEW!	Symtab (general-purpose symbol table construction and management package)	$60
	PTree (general-purpose parse tree construction and management package)	$60
NEW!	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	Polyglot TSR Package (includes reminder, bookmark, virus catcher, cache manager, & speech generator)	$50
	HELP! (pop-up help system builder)	$50
	Multi-User BBS (chat, mail, menus, sysop displays; uses Galacticomm modem card; Version 4.1.1)	$50
	Make (macros, all languages, built-in rules)	$50
	Coder's Prolog (inference engine for use with C programs)	$45
	Virtual Memory Manager by Blake McBride (LRU pager, dynamic swap file, image save/restore)	$40
	Heap I/O (treat all or part of a disk file as heap storage)	$40
	Biggerstaff's System Tools (multi-tasking window manager kit)	$40
	OOPS (collection of handy C++ classes by Keith Gorlen of NIH; Version 2.2)	$35
	Bison & PREP (YACC workalike parser generator & attribute grammar preprocessor; now includes documentation)	$35
	PC-XINU (Comer's XINU operating system for PC)	$35
	CLIPS (rule-based expert system generator, Version 4.2)	$35
	Tiny Curses (Berkeley curses package)	$35
	Polyglot RAM Disk (change disk size on the fly; includes utilities)	$30
	SP (spelling checker with dictionary and maintenance tools)	$30
	Clisp (Lisp interpreter with extensive internals documentation)	$30
	Translate Rules to C (YACC-like function generator for rule-based systems)	$30
	6-Pack of Editors (six public domain editors for use, study & hacking)	$30
	Crunch Pack (14 file compression & expansion programs)	$30
	Pascal P-Code Compiler & Interpreter or Pascal-to-C Translator (Wirth standard Pascal)	$25
	ICON (string and list processing language, Version 7.5)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX; Version 1.1)	$25
	LEX (lexical analyzer generator; an oldie but a goodie)	$25
	AutoTrace (program tracer and memory trasher catcher)	$25
	Data Handling Utilities in C (data entry, validation & display; specify Turbo C or Microsoft)	$25
	Arrays for C (macro package to ease handling of arrays)	$25
	A68 (68000 cross-assembler)	$20
	List-Pac (C functions for lists, stacks, and queues)	$20
	XLT Macro Processor (general purpose text translator)	$20
	C/reativity (Eliza-based notetaker)	$15
	Data	
Updated!	GenBank DNA Sequences (Release 59.0; includes demo disk of Pearson FAST/A programs)	$150
	Protein Sequences (over 10,000 sequences; includes demo disk of Pearson FAST/A programs)	$60
NEW!	Smithsonian Astronomical Observatory Subset (right ascension, declination, & magnitude of 258,997 stars)	$60
	Dictionary Words (234,932 words in alphabetical order, no definitions)	$60
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
	KST Fonts (13,200 characters in 139 mixed fonts: specify TEX or bitmap format)	$30
NEW!	USNO Interactive Computer Ephemeris (high-precision moon, sun, planet & star positions)	$30
	King James Bible (Old and New Testaments)	$25
	NBS Hershey Fonts (1,377 stroke characters in 14 fonts)	$15
	U. S. Map (15,701 points of state boundaries)	$15

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3409 USA

acw!info@uunet.uu.net

Voice: (512) 258-0785
BBS: (512) 258-8891
FAX: (512) 258-1342

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

Number 30 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Vernon code, continued

3

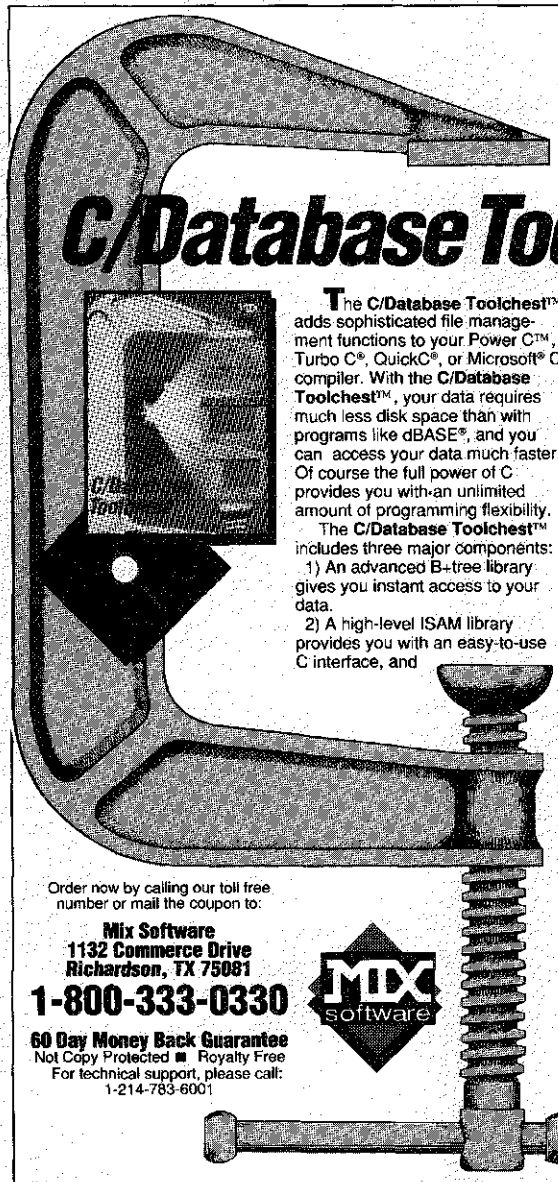
```
        DisplayAttr() { attribute = normal(); }
        ~DisplayAttr() { }
};

class Display : public Rectangle {

protected:
        Point           cursor;
        Mouse   *       mouse;
        DisplayAttr     defAttr;
        unsigned        color;
        unsigned        snow;
        static unsigned videoSeg;
        static char     init;
public:
        Display(const Mouse *);
        ~Display();

        virtual void    clear();
        void            operator()(unsigned, unsigned);
        void            operator()(Point & p) { cursor = p; }
        Point &         operator()() { return cursor; }
        void            cursorOff();
        void            cursorOn();
        unsigned        hasColor()      { return color; }
        void            defaultAttr(DisplayAttr & a) { defAttr = a; }
        virtual void    putString(Point &, const unsigned char *,
                                DisplayAttr &);
        virtual void    putChar(Point &, unsigned char, DisplayAttr &);
        Display &       operator<<(const unsigned char * s) {
                        putString(cursor, s, defAttr);
                        return *this;
        }
        Display &       operator<<(unsigned char c) {
                        putChar(cursor, c, defAttr);
                        return *this;
        }
        virtual void    draw();
};

#endif // __DISPLAY_CLASS__

// display.cpp  the Display class methods
//
// © Aspen Scientific 1989. All Rights Reserved.
// Author: Vaughn Vernon

#include "display.cls"

// for Microsoft C, not Zortech
#ifndef __ZTC__
# include <memory.h>
# include <conio.h>
#else
static void movedata(unsigned, unsigned, unsigned, unsigned, unsigned);
#endif

// ****************************************************************
// ** the Display methods in this file are implemented for MS-DOS
// ** in a text display (non-graphics) environment.  it uses
// ** 8086 interrupts and the Microsoft C int86() function, as
// ** well as far pointers to the video memory map segment.
// ****************************************************************

#include <dos.h>

static union REGS vRegs;
static union REGS videoSaveMode;
static const unsigned monoSeg  = 0xb000;
static const unsigned colorSeg = 0xb800;

#if defined(M_I86SM) || defined(M_I86MM)
static unsigned ds;
#endif
```

4

```
static unsigned map[80];

Display::Display(const Mouse * mou) : cursor(0,0)
{
        if (init == 1)     return;

        mouse = mou;

        // for small and medium models (small data), need data seg
        struct SREGS segs;
        segread(&segs);
        ds = segs.ds;

        // find out the current video mode, save it, then set the new mode if necessary.

        vRegs.h.ah = 0x0f;
        int86(0x10, &vRegs, &vRegs);
        videoSaveMode = vRegs;

        // return of 7 in AL register is mono
        if (vRegs.h.al == 7) {
                videoSeg = monoSeg;
                color = 0;
                snow  = 0;
        }
        else {

                videoSeg = colorSeg;
                color = 1;

                if (vRegs.h.al != 3) {
                        vRegs.h.ah = 0;
                        vRegs.h.al = 3;
                        int86(0x10, &vRegs, &vRegs);
                }

                // check for a snowing display; an EGA
                // or VGA is automatic no-snow.

                unsigned EGAMemory, EGAMode;

                vRegs.h.ah = 0x12;
                vRegs.h.bl = 0x10;
                int86(0x10, &vRegs, &vRegs);

                EGAMemory = vRegs.h.bl;
                EGAMode   = vRegs.h.bh;

                // if the returned memory is out of range, or
                // the ega/vga card is not attached to the color
                // display (EGAMode != 0), assume snow.

                if (EGAMemory < 0 || EGAMemory > 3 || EGAMode != 0)      snow = 1;
                else     snow = 0;
        }

        // set the Display size
        origin(Point(0,0));
        corner(Point(24,79));

        clear();
        draw();
        init=1;
}

Display::~Display()
{
        clear();
        cursor(0,0);
        draw();
        vRegs = videoSaveMode;
        int86(0x10, &vRegs, &vRegs);
}
```

Code continues . . .

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Turn your favorite C compiler into a powerful database manager with the

C/Database Toolchest

The **C/Database Toolchest**™ adds sophisticated file management functions to your Power C™, Turbo C®, QuickC®, or Microsoft® C compiler. With the **C/Database Toolchest**™, your data requires much less disk space than with programs like dBASE®, and you can access your data much faster. Of course the full power of C provides you with an unlimited amount of programming flexibility.

The **C/Database Toolchest**™ includes three major components:

1) An advanced B+tree library gives you instant access to your data.

2) A high-level ISAM library provides you with an easy-to-use C interface, and

3) A complete database manager (with C source code included) shows you how to create impressive applications.

You also receive a comprehensive 350 page manual and a utility for converting dBASE® files.

The **C/Database Toolchest**™ supports features that you'd expect to find only in products costing ten times as much. Advanced features include variable length records, variable length keys, multiple keys per index, and multiple indexes stored in a single file. Your data files can contain an unlimited number of records, and each record can be as large as 32K bytes in length.

About the only thing that the **C/Database Toolchest**™ doesn't do is cost you a lot of money. We've kept our price low so you can manage your budget as easily as your data.

Now Only $19.95!

Order now by calling our toll free number or mail the coupon to:

Mix Software
1132 Commerce Drive
Richardson, TX 75081

1-800-333-0330

60 Day Money Back Guarantee
Not Copy Protected ■ Royalty Free
For technical support, please call:
1-214-783-6001

MIX software

Order Coupon

Name_____
Street_____
City_____
State_____ Zip_____
Telephone_____
Paying By___ Money Order___ Check
___ Visa ___ MC ___ AX ___ Disc.
Card#_____
Exp. Date_____
Disk Size___ 5¼" ___ 3½"

Qty.	Product	Price	Subtotal
___	C/Database Toolchest	$19.95	___
___	C/Database Library Source (incl. B+tree & ISAM library source code)	$10.00	___
	Add Shipping ($5 USA, $20 Foreign)		___
	Texas Residents Add 8% Sales Tax		___
	Total Amount of Your Order		___

J

C/Database Toolchest and Power C are trademarks of Mix Software. QuickC and Microsoft C are registered trademarks of Microsoft. Turbo C is a registered trademark of Borland. dBASE is a registered trademark of Ashton-Tate.

Number **266** on the Reader Service Card

DOS & OS/2
Free Demo Diskette Available

Q. How many programmers does it take to maintain a MAKE dependency file?

A. NONE! *If you use* VersiMAKE™

VersiMAKE™ is a full-featured MAKE utility that includes:

■ **Dependency Generation**
Derives your system's dependencies, through analysis of its C and MASM source files. Say goodbye to manual maintenance of MAKE dependency files!

■ **Wild Card File Name Matching**
Analyzes an entire collection of source files with a single statement.

■ **Nested Include Files**
Handles standard C and MASM "include" conventions, and the INCLUDE environment variable.

■ **Powerful Macro Facilities**
Built-in macros, user-defined macros, and environment variables.

■ **Analytical Reports**
Shows the entire Include file hierarchy for each source file analyzed, and all of the parent source files for each Include file.

Q. How many programmers does it take to trace a symbol thru your system?

A. ONE! *If you use* VersiCREF™

VersiCREF™ is a unique utility that creates a Master Cross-Reference of your entire system.

■ **Multi-Lingual**
Handles C, assembler, or both.

■ **Flexible**
File names with line numbers, or file names alone. Global and local symbols, or globals alone.

■ **Powerful**
Easily handles systems containing 100 source files or more.

VersiMAKE™	$125
VersiCREF™	$75
Both	$150

(Outside U.S., add $5 for shipping and handling)

800-334-4096
(In NJ, 609-871-0202)
MC/VISA/AMEX accepted

SUMMIT INFORMATION SYSTEMS, INC.
73 East Lane, Willingboro, NJ 08046

Number **158** on the Reader Service Card

CHANGE OF ADDRESS?

Please send us your new address along with your previous ZIP code, using this handy form— Thanks.

NAME_____
COMPANY_____
ADDRESS_____
CITY_____ STATE_____
ZIP CODE_____ OLD ZIP CODE—_____
TELEPHONE_____

Please mail to:
Programmer's Journal
P. O. Box 30160
Eugene, OR 97403

Or call toll free
800-234-0386

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Vernon code, continued

5

```
void
Display::clear()
{
        Point size = extent();

        mouse->hide();

        vRegs.h.dh = size.x() + 1;
        vRegs.h.dl = size.y() + 1;
        vRegs.h.bh = defAttr.get() >> 8;
        vRegs.x.cx = 0;
        vRegs.h.al = 0;
        vRegs.h.ah = 6;
        int86(0x10, &vRegs, &vRegs);

        mouse->show();
}

void
Display::operator()(unsigned row, unsigned col)
{
        Point top, bot;

        top = origin();
        bot = corner();

        if (row < top.x() || row > bot.x() || col < top.y() || col > bot.y())
                return;

        cursor(int(row), int(col));        // move cursor
        draw();
}

void
Display::cursorOff()
{
        vRegs.x.cx = 0x2000;
        vRegs.h.ah = 1;
        int86(0x10, &vRegs, &vRegs);
}

void
Display::cursorOn()
{
        if (hasColor())         vRegs.x.cx = 0x0607;
        else                    vRegs.x.cx = 0x0b0c;
        vRegs.h.ah = 1;
        int86(0x10, &vRegs, &vRegs);
}

void
Display::putString(Point & p, const unsigned char * s, DisplayAttr & a)
{
        cursor = p;

        register unsigned c = cursor.y();
        register unsigned len=0;
        unsigned *mp = map;
        for ( ; c <= width() && *s; ++c, ++len)
                *mp++ = (unsigned)*s++ | a.get();

        mouse->hide();

        if (snow)               // protect against flicker
                while ((inp(0x03da) & 8) == 0)


#if defined(M_I86SM) || defined(M_I86MM)
        movedata(ds, (unsigned)map,
#else
        movedata(FP_SEG(map), FP_OFF(map),
#endif
                videoSeg, cursor.x() * 160 + cursor.y() * 2, len * 2);
```

6

```
        mouse->show();
        cursor(cursor.x(), c);
}

void
Display::putChar(Point & p, unsigned char c, DisplayAttr & a)
{
        cursor = p;

        *map = c | a.get();

        mouse->hide();

        // protect against flicker
        if (snow)
                while ((inp(0x03da) & 8) == 0)
                        ;

#if defined(M_I86SM) || defined(M_I86MM)    movedata(ds, (unsigned)map,
#else
        movedata(FP_SEG(map), FP_OFF(map),
#endif
                videoSeg, cursor.x() * 160 + cursor.y() * 2, 2);

        mouse->show();

        if (cursor.y() != width())        cursor(cursor.x(), cursor.y() + 1);
}

// just update cursor
void
Display::draw()
{
        vRegs.h.dh = cursor.x();
        vRegs.h.dl = cursor.y();
        vRegs.h.bh = 0;
        vRegs.h.ah = 2;
        int86(0x10, &vRegs, &vRegs);
}

#ifdef __ZTC__

static void
movedata(unsigned sSeg, unsigned sOff, unsigned dSeg, unsigned dOff, unsigned n)
{
        unsigned far *src, far *dest;

        src  = MK_FP(sSeg, sOff);
        dest = MK_FP(dSeg, dOff);

        n >>= 1;        // divide by 2

        for (register int i=0; i < n; ++i)        *dest++ = *src++;
}

#endif
```

LISTING 3

```
// mouse.cls    a C++ Mouse class -- for DOS.
//
// © Aspen Scientific 1989. All Rights Reserved.
// Author: Vaughn Vernon

#ifndef __MOUSE_CLASS__

# define __MOUSE_CLASS__        1
# include "point.cls"

class Mouse {

protected:      // used by derived class
        int     installed;
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    int     numButtons;

public:
            Mouse();
            ~Mouse();

            // tell user how many buttons the mouse has
            int         buttons() { return numButtons; }
            virtual void    hide();
            virtual void    show();
            virtual int getEvent(Point &,unsigned & b1,unsigned & b2,unsigned & b3);
            virtual void    setPosition(Point &);
            virtual void    flush();

            // this method is strictly system dependent. To use this, a class must
            // be derived and the  virtual function replaced. The argument points
            // to an implementation dependent struct of some
            // sort which defines the type of mouse-pointer needed.

            virtual void makePointer(const void *) { }

};

#endif // __MOUSE_CLASS__

// mouse.cpp    implements methods for the Mouse class. see "mouse.cls".
//
// © Aspen Scientific 1989. All Rights Reserved.
// Author: Vaughn Vernon

#include "mouse.cls"

// ***************************************************
// ** the Mouse methods in this file are implemented for MS-DOS
// ** in a text display (non-graphics) environment.  it uses
// ** 8086 interrupts and the Microsoft C int86() function.
// ***************************************************

#include <dos.h>

static union REGS mRegs;
static union REGS holdforEvent;

Mouse::Mouse()
{
        if (peek(0,0x33*4+2) == 0)
                installed = 0;
        else {
                mRegs.x.ax = 0;
                int86(0x33, &mRegs, &mRegs);

                // AX == 0 means mouse not installed
                if (mRegs.x.ax == 0)
                        installed = 0;
                else {
                        installed = 1;
                        numButtons  = mRegs.x.bx;

                        // set the text software cursor and make all characters
                                pointed to reverse-video.
                        mRegs.x.ax = 10;
                        mRegs.x.bx = 0;
                        mRegs.x.cx = 0x00ff;
                        mRegs.x.dx = 0x7000;
                        int86(0x33, &mRegs, &mRegs);

                        show();
                        flush();
                }
        }
}

Mouse::~Mouse()
{
        if (installed) {
                // under ms-mouse and dos, re-initializing shuts things down
                mRegs.x.ax = 0;
```

Code continues . . .

CONTINUED BY POPULAR DEMAND!

All 6 issues from 1988— featuring the highly acclaimed "On Graphics" series by Michael Abrash **plus** the VGA code disk.

Special Price—

only **$35.00**

A savings of **$17.00** off the regular price

To order call toll free

800-234-0383

Or see our ad on page 88

FULL AT&T C++: ANNOUNCING VERSION 1.2

Guidelines announces its port of **version 1.2** of AT&T's C++ translator. As an object-oriented language C++ includes: classes, inheritance, member functions, constructors and destructors, data hiding, and data abstraction. "Object-oriented" means that C++ code is more readable, more reliable and more reusable. And that means faster development, easier maintenance, and the ability to handle more complex projects. C++ is **Bell Labs' answer to Ada and Modula 2.** C++ will more than pay for itself in saved development time on your next project.

C++

from GUIDELINES for the IBM PC: $295

Requires IBM PC/XT/AT or compatible with 640K and a hard disk.
Note: C++ is a *translator,* and requires the use of Microsoft C 3.0 or later.

Here is what you get for $295:

- The full AT&T v1.2 C++ translator.
- Libraries for stream I/O and complex math.
- **The C++ Programming Language,** the definitive 327-page tutorial and description by Bjarne Stroustrup, designer of C++.
- Sample programs written in C++.
- Improved installation guide and documentation.
- 30-day money-back guarantee.

NOW AVAILABLE FOR UNIX V/386 – $495

To Order:
send check or money order to:

GUIDELINES SOFTWARE, INC.
P. O. Box 749, Dept. PJ
Orinda, CA 94563

To order with VISA or MC, phone (415) 254-9183.
(CA residents add 6% tax.)

C++ is ported to the PC by Guidelines under license from AT&T.
Call or write for a free C++ information package.

Number 204 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Vernon code, continued

8

```
                int86(0x33, &mRegs, &mRegs);
        }
}

void
Mouse::hide()
{
        if (installed) {
                mRegs.x.ax = 2;
                int86(0x33, &mRegs, &mRegs);
        }
}

void
Mouse::show()
{
        if (installed) {
                mRegs.x.ax = 1;
                int86(0x33, &mRegs, &mRegs);
        }
}

int
Mouse::getEvent(Point & p, unsigned & b1, unsigned & b2, unsigned & b3)
{
        if (!installed)         return 0;

        mRegs.x.ax = 3;
        int86(0x33, &mRegs, &mRegs);

        if (mRegs.x.bx == holdForEvent.x.bx &&
            mRegs.x.cx == holdForEvent.x.cx &&
            mRegs.x.dx == holdForEvent.x.dx)
                return 0;
        holdForEvent = mRegs;

        // vertical and horizontal cursor poition
        p(mRegs.x.dx / 8, mRegs.x.cx / 8);

        // are the buttons down?
        b1 = (mRegs.x.bx & 0x01) ? 1:0;
        b2 = (mRegs.x.bx & 0x02) ? 1:0;
        b3 = (mRegs.x.bx & 0x04) ? 1:0;

        return 1;
}

void
Mouse::setPosition(Point & p)
{
        if (installed) {
                mRegs.x.ax = 4;
                mRegs.x.cx = p.y();
                mRegs.x.dx = p.x();
                int86(0x33, &mRegs, &mRegs);
        }
}

void
Mouse::flush()
{
        Point p;
        unsigned b1, b2, b3;

        while (getEvent(p, b1, b2, b3))
}
```

LISTING 4

```
// keyboard.cls a Keyboard class for C++.
//
// this file implements a Keyboard class.  it hides system details from
// the class user.
```

9

```
//
// © Aspen Scientific 1989. All Rights Reserved.
// Author: Vaughn Vernon

#ifndef __KEYBOARD_CLASS__

# define __KEYBOARD_CLASS__      1
# define extMask         0x8000

class Keyboard {

protected:
        int     keyI;           // saves latest input
public:
        Keyboard()                          { keyI = -1; }
        ~Keyboard()                         { }

        // methods
        int     input(int wait =1);
        int     value()                     { return keyI; }
        int     operator==(int keyComp)     { return (keyI == keyComp); }
        Keyboard & operator>>(int & k)      { k = input(1); return *this; }
        void    flush()                     { while (input(0) != -1) ; }
};

class KeyTypes {
public:
        KeyTypes()      { }
        ~KeyTypes()     { }

        // these methods are used to compare input to
        int     None()          { return (-1); }
        int     Escape()        { return 27; }
        int     SpaceBar()      { return ' '; }
        int     Tab()           { return '\t'; }
        int     BackTab()       { return 15; }

        int     Printable(int key)      {
                        return (key >= 32 && key <= 127);
        }
        int     ASCII(int key)  {
                        return (key >= 0 && key <= 255);
        }
        int     Ctrl(char c)    { return (c & 037); }

        int     Home()          { return (71 | extMask); }
        int     End()           { return (79 | extMask); }
        int     PgUp()          { return (73 | extMask); }
        int     PgDn()          { return (81 | extMask); }
        int     Left()          { return (75 | extMask); }
        int     Right()         { return (77 | extMask); }
        int     Up()            { return (72 | extMask); }
        int     Down()          { return (80 | extMask); }

        int     F1()            { return (59 | extMask); }
        int     F2()            { return (60 | extMask); }
        int     F3()            { return (61 | extMask); }
        int     F4()            { return (62 | extMask); }
        int     F5()            { return (63 | extMask); }
        int     F6()            { return (64 | extMask); }
        int     F7()            { return (65 | extMask); }
        int     F8()            { return (66 | extMask); }
        int     F9()            { return (67 | extMask); }
        int     F10()           { return (68 | extMask); }
};

#endif // __KEYBOARD_CLASS__

// keyboard.cpp Keyboard class non-inline methods.
//
// © Aspen Scientific 1989. All Rights Reserved.
// Author: Vaughn Vernon

#include "keyboard.cls"
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ Code Is Available!

On disk from:

Programmer's Journal
P. O. Box 30160
Eugene, OR 97403
800-234-0386
503-747-0800 in Oregon

All Code Disks $10.00 each.
To order see ad on page 88.

PJ Code DiskScriptions are available for only $39.95.
To order see ad on page 8.

On BBS from:

Mnematics Videotex
P.O. Box 19
Sparkill, NY 10976-0019
(Voice) 800-322-3633
(Voice) 914-365-0184 in N.Y.
Mention *PJ* when you call.

FIDO/OPUS Network
Blazer Net 152 Node 9
503-343-9724

Generation Five BBS
(Voice) 301-495-6690
(Data) 301-495-2932
Accessible via PC Pursuit, Washington D.C. From Files submenu, select "P" for Personal Downloads.

PCS-49 on CompuServe
Section 9
Sponsored by:
MISOSYS, Inc.
P.O. Box 239
Sterling, VA 22170
(Voice) 800-MISOSYS

VIDEO TAPE

Understanding C.

3 volumes, 6 hours, covers every nook and cranny of ANSI C. Loaded with examples! Results Guaranteed! Comes with reference book. You'll learn techniques used by experienced C programmers. $59.95 post paid.

Send check or money order to:
AUTOMATION EDUCATION
2309 Royce Drive
Arlington, TX 76016-1232
(817) 496-8747

Number **207** on the Reader Service Card

10

```
#include <dos.h>
#include <conio.h>

static union REGS kRegs;

int
Keyboard::input(int wait)
{
    if (!wait && !kbhit())
        return (keyI = -1);
    kRegs.h.ah = 0;
    int86(0x16, &kRegs, &kRegs);

    if (kRegs.h.al != 0)    keyI = kRegs.h.al;    // regular key
    else
        keyI = kRegs.h.ah | extMask;    // extended key
    return keyI;
}
```

```
LISTING 5
// dots.cpp      draw dots on the display with mouse.  button 1 (left)
//                 puts dots on, and buttons 2&3 (middle/right, right)
//                 take dots off.
//
// © Aspen Scientific 1989.  All Rights Reserved.
// author: Vaughn Vernon

#include "point.cls"
#include "display.cls"
#include "mouse.cls"
#include "keyboard.cls"

int
main()
{
    Mouse mouse;
    Display display(&mouse);
    Keyboard kb;
    KeyTypes ktype;
    DisplayAttr attr;

    if (display.hasColor())    attr.set( attr.foreWhite() | attr.backBlue() );
    else   attr.set( attr.reverse() );

    display.putString(Point(0, 0), " Press [Esc] to quit. ", attr);

    if (display.hasColor())    attr.set( attr.foreYellow() | attr.backBlack() );
    else   attr.set( attr.normal() );

    display.cursorOff();

    Point mpoint;
    unsigned b1, b2, b3;

    while( 1 ) {
        if (mouse.getEvent(mpoint, b1, b2, b3) != 0) {
            if (b1)    display.putChar(mpoint, '.', attr);
            else if (b2 || b3)
                display.putChar(mpoint, ' ', attr);
            display.draw();
        } else if (kb.input(0) == ktype.Escape())
            break;
    }
    display.cursorOn();
}
```


PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

dBASE C Libraries

CodeBase (Nov. 1988 version)
Sequiter Software
9315 70 Avenue
Edmonton, Alberta
Canada T6E OT8
403-439-8171

Sometimes a database task will require more algorithmic power than is easily available in the dBASE language, or the program must be supported on a platform different from DOS. Sequiter Software has released CodeBase ($194), a C library for database management. The CodeBase library can be used to create stand-alone programs that are compatible with dBASE or Clipper DBF, NDX, and NTX files. The CodeBase library consists of 165 routines coded in C (260K of source code) that generally correspond directly with the dBASE commands and functions they emulate. CodeBase is provided as source code along with object libraries for Turbo C (large model) and QuickC/Microsoft C (medium model). Both single and multi-user applications are supported. The CodeBase routines have built-in support for file and record locking for network applications. Equates in the code support some OS/2 features. The routines can be divided into the functional categories of database files, index files, field access, memo fields, GET and SAY I/O routines, expression evaluation, memory handling, conversion, utility, and extended (filtering and relations) routines.

The database and index file routines correspond most directly to the common dBASE commands with names using D3 and I3 as the prefix (e.g., d3use, d3select, d3skip, i3index). These routines incorporate a useful safety flag to prevent overwriting an existing file. The field routines access dBASE files as an equivalent C structure, by field number, or by direct access. Support is provided for C routines similar to the dBASE SAY, GET, and READ commands. Routines allow conversion between database fields and special C variable types and structures (dates, doubles, and so on). A set of memory routines are used by the file routines and also provide allocation and freeing of small chunks of memory with little overhead. The expression evaluator supports a very limited subset of dBASE functions and is designed primarily for use with the CodeBase index commands. For speed, CodeBase evaluates expressions in a two-step process—the expression is first pseudocompiled and then evaluated. One of the extended routines even supports a semblance of the dBASE BROWSE command.

Although only a subset of a dBASE parser is provided, one could argue that these functions should be coded directly in C. Compiled C expression code executes an order of magnitude faster than interpreted dBASE expressions. Most dBASE high-level commands are fully supported. The overall design emulates dBASE in function naming and features. The programs and routines I tested provided speed faster than dBASE III but not up to the speed of, say, FoxBase for indexing. Executable files were generally small and are supposed to average about half the size of equivalent Clipper EXE files.

CodeBase comes with a 125-page user's guide including a table of contents and an index. The user's guide could benefit from more tutorial materials and programming examples. Appendices cover errors, expressions, compatibility, functions, and library building. An example program is provided to illustrate each CodeBase function operation and act as a tutorial. Executable programs developed with CodeBase may be distributed at no additional charge. Although the documentation could be improved, CodeBase appears to be a solid product. It supports both dBASE and Clipper.

Apex Database Library (ADL) v1.0
Apex Software
4516 Henry Street, Suite 308
Pittsburgh, PA 15213-3785
412-681-4343

The Apex Database Library (ADL) is another C library for dBASE file management. ADL provides a set of C functions fully compatible with dBASE III. ADL is provided as object libraries in five memory models for both Microsoft C (4.0 and 5.1) and TurboC (2.0) for $395. The source code is available for an additional $400. Apex Software already has developed one product, Frontrunner, for Ashton-Tate and is now developing the compiler to be included with dBASE IV. The design of this C library package is based on a standard C file I/O model rather than the dBASE language model. This may provide an easier learning path for C programmers than CodeBase's dBASE syntax. ADL provides complete functions for data, index, and memo file management compatible with dBASE. ADL also supports configurable caching buffers for database, index, and memo files. But the real strength of this package is the almost complete dBASE expression parser support. Where CodeBase supports maybe 12 dBASE functions in expressions, ADL supports 60 of the equivalent dBASE parser features. The current version supports only single-user applications; however, a network version supporting file and record locking will soon be released.

The dBASE parser is provided in three versions: a full parser, a regular parser (70% of dBASE functions), and a field-only parser (supports single fields only). Using one of the subset parsers reduces memory and code-size requirements. The current version of ADL provides no SAY or GET I/O routines comparable with those in dBASE. One must presume that the C developer will use a screen interface of his or her own choosing. Records can be accessed by field access, C structure access, or direct access. ADL comes with a useful utility, CSTRUCT, in source code that generates a suitable C structure from an existing DBF file. Memo fields are well supported. Exception handling is a bit novel: Although a C function typically would return an error code, ADL uses a scheme of registered exception-handler routines to simplify error checking. A stack of 32 error handlers may be supported. If an error occurs, the appropriate error-handling routine is called directly.

ADL functions use Pascal calling conventions. The executable code appeared to execute faster than dBASE for the operations I tested but still not up to the speed of FoxBase for indexing. ADL was generally faster than CodeBase, but if disk-caching software (such as IBMCACHE or Compaq's CACHE) were used, these differences probably would be minimal. Although the dBASE parser is very complete, some convenient high-level commands are not included. For example, to index or re-index a database requires setting up a loop and processing each record individually. No safety flags are provided to prevent writing over an existing database, index, or memo file. Although multi-user support is omitted in this release, I have reviewed the manual revisions to support networking—the proposed file-locking and record-locking mechanisms are very flexible.

ADL comes with an excellent 187-page manual with a table of contents and an index. About the first third of the manual is a tutorial, which uses illustrative code examples to explain ADL usage and covers installation; database, index, and memo files; caching mechanisms; the expression parser/interpreter; and exception handling. The remainder of the manual is a detailed reference section with a single function per page and a usage example for each function. Appendices cover error messages and even dBASE file formats. Executable programs using ADL may be distributed at no charge. If you need a complete dBASE parser, then ADL provides this plus a solid and fast dBASE C function file library.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Software Tools

Optlib and Optlib/2
SLR Systems
1622 N. Main St.
Butler, PA 16001
800-833-3061

SLR systems has released OPTLIB and OPTLIB/2 ($49 each), high-performance library utilities for MS-DOS and OS/2, respectively. While the two librarian products offer the same features, the DOS product has been optimized for speed in the real mode, and the OS/2 product is optimized for use under OS/2. These utilities can create and maintain libraries for object modules produced by assemblers and by high-level language compilers that produce standard OMF object files. SLR is the developer of OPTASM, the fastest macro assembler currently available under DOS (see *P.J* 6.3).

The first thing you'll notice about OPTLIB is that it's very fast. In my timing tests, I found it outpaced the latest Microsoft LIB under DOS by three to ten times. OPTLIB is compatible with other librarians but adds some very useful extensions and features. OPTLIB has the ability to retain the original time/date stamp of the object file on extracted modules. This allows a module in the library to be refreshed (replaced) if the OBJ file's date is more recent, as with Make utilities. An option allows renaming public or external symbols in a library. The excellent report-listing options include an indexed listing by modules, a map option showing modules and named entry points, a complete cross-reference–generation capability for each symbol in the library, and a statistics summary at the completion of the run. OPTLIB also supports an option to correct **checksum** errors in a module, and it can handle multidefined symbols.

The OPTLIB librarians are user configurable for default options with a supplied CONFIG utility. OPTLIB will also use a dedicated environment variable for getting user options. Finally, a long list of command line options are supported. Each version of OPTLIB comes with an excellent 50-plus–page manual that includes a decent index. If you use a librarian, OPTLIB is a fast and polished product.

Inside!
Paradigm Systems Inc.
P.O. Box 152
Milford, MA 01757
800-537-5043

Paradigm Systems has released version 2.0 of its Inside! family of performance-analysis

POWER GRAPHICS FOR YOUR PC!

MetaWINDOW

NO ROYALTIES!

PC TECH JOURNAL
Product of the Month
"Awarded for excellence in product design and development"

Developing a graphics application? Move to the peak of performance and graphics functionality with MetaWINDOW! Built on state of the art "object oriented" display flexibility, now you can take advantage of powerful advanced graphics capabilities like those used on the Macintosh, Sun and Apollo workstations. Using MetaWINDOW you can break the bottleneck between limited function low-level graphic primitive libraries and slow, bulky hard-coded window managers.

MetaWINDOW provides an advanced set of graphic drawing functions, dynamic runtime support for a full range of graphic peripherals, plus unmatched performance for designing multi-window desktop applications.

Awarded Top Honors!

PC Tech Journal, Jul '85
Product of the Month - *"... a technological tour de force for fast PC graphics."*
"... one high-powered piece of artillery."

Computer Language, Oct '87 -
"... surprisingly powerful for the cost."
*Independently benchmarked the fastest of all graphics systems reviewed - faster than Halo, over 3 times faster than GSS*CGI.*

Turbo Technix, Feb '88
"... Customer support is excellent. I've called Metagraphics several times and have gotten prompt, accurate help each time. Metagraphics has a winner with TurboWINDOW."

Superior Functionality!

- Macintosh QuickDraw function superset - *simplifies porting PC & Mac applications.*
- PostScript & X-Window-compatible drawing attributes - the defacto graphics *standard for PC X-Window servers.*
- Supports both bitmap and outline fonts - *dynamically face fonts for bold, italic, underline or strike-out effects.*
- Full 16 function "rasterOp" flexibility - *easily write, erase, rubberband or drag: lines, text, icons, bit images and complex objects.*
- Dynamic recognition and run-time support for 80+ popular graphics cards - *eliminates multitudes of clumsy device drivers and bothersome configuration sessions - true device independence at last!*
- Virtual images - *create and manipulate graphic images in local or expanded memory up to 8 megabytes in size.*
- *... and the list goes on!*

Rated #1 by Users!

J. Kayser, CMSI -
"MetaWINDOW is the best graphics package I've seen. Since I bought my PC, I've spent over $2500 on software. I've bought bad software, mediocre software, and good software. There's not much software I call excellent, but MetaWINDOW is excellent!"

David Gray, LUCAS LABS -
"I would also like to take this time to compliment you on providing such a fantastic graphics package. We have used several packages previously including Turbo graphics and Halo graphics and found none that compare to the versatile, high performance graphics package that your company offers. We have also found your technical support staff to be one of the best we have ever dealt with."

John Weaver, Chairperson
Department of Mathematical Sciences
WEST CHESTER UNIVERSITY -
"We have been using MetaWINDOW for our graphics instruction for the past several semesters and are pleased with the package. This past summer we upgraded one system to MetaWINDOW version 3 and were so impressed that we want to upgrade all systems prior to the Spring semester. This is a first class package and I think it is the best for teaching computer graphics."

Reach For The Future!

MetaWINDOW - $195
Advanced Graphics Development Toolkit
MetaWINDOW comes complete with language bindings for 20 popular C, Pascal and Fortran compilers, plus dynamic runtime support for over 80 graphics adaptors and input devices.
4 disks, 3 260 page manuals

TurboWINDOW/C - $95
TurboWINDOW/Pascal - $95
QuickWINDOW/C - $95
All the features of MetaWINDOW for Borland Turbo C, Turbo Pascal or Microsoft C.

NEW - FontWINDOW - $95
MetaWINDOW font/icon editor plus 100 fonts.

NEW - MetaWINDOW/Premium - $495
Power graphics for new high-res, high performance TI34010 based graphics cards!
(Licensing required for redistribution of MetaWINDOW/Premium)

30 Day Money Back Guarantee
ORDER TODAY! 800-332-1550
For information or in CA call 408-438-1550

METAGRAPHICS
SOFTWARE CORPORATION
269 Mount Hermon Road
Scotts Valley, CA 95066

Number 143 on the Reader Service Card

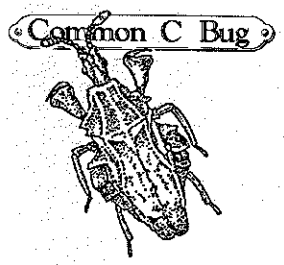
PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

T-SHIRTS

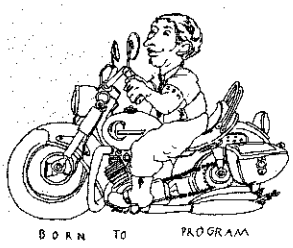
$12.00 each -- includes shipping & handling

Multi-color imprints on the highest quality 50/50 cotton T-shirts.

SIZES S, M, L, XL

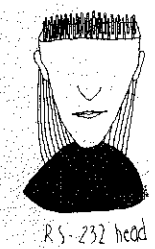


COMMON C BUG



BORN TO PROGRAM

PARALLEL PORTS



RS-232 HEAD

BUGS RULE 4 color imprint on a white shirt

PJ ISSUE SPECIAL!

Only $35.00! Save 33% off the regular price!

CONTINUED BY POPULAR DEMAND!

The VGA Graphics Express

You get all six 1988 back issues plus the VGA disk containing the code from all of Michael Abrash's 1988 and 1989 "On Graphics" VGA articles at a savings of $17.00 off the regular price.

(Offer expires June 30, 1989.)

BACK ISSUES

$7.00 each PREPAID ONLY

A few of the Back Issues currently available

Vol. 5.3: Starting your own software house—publicity; UNIX books, part 2; Consulting—merger mania; EGA data control; Modula-2 for Pascal lovers; optimizing Turbo Pascal; ANSI C language standard; coping with ANSI C and forcing file updates; formatting BASIC output with buffered PRINT USING.

Vol. 5.4: Starting your own software house—technical support; UNIX books, part 3; Consulting—productivity and effectiveness; EGA set/reset capabilities; ANSI C language standard; character tables and string functions in C; MWWIND—a menu with Windows utility; variable passing and hiding in QuickBASIC 2 subprograms.

Vol. 5.5: Intel's 80386 vs. OS/2; interview with Metagraphics President Jack Davis; Consulting—making computer tools pay for themselves; ASM—the invisible CapsLock; TSRs using expanded memory with Turbo Pascal; note on optimizing Turbo Pascal; proposed ANSI C Language Standard; printer output from Turbo C programs; match-up of Turbo Basic 1.0 and QuickBASIC 3.0.

Vol. 5.6: Multics, OS/2, paging, and segmentation; The Usenix conference and other news; how to get rich in consulting; Shell in Marshal Pascal; Turbo Pascal 4.0 preview; handling fixed-length character arrays in C; the proposed ANSI C language standard.

Vol. 6.1: Consulting is an attitude; write mode 3 of the VGA; the LIM 4.0 expanded memory specification; environment variables in Turbo Pascal 4.0; a timer program in C; porting mainframe software to the PC—FORTRAN compilers under DOS and the Definicon 32-bit coprocessing engines.

Vol. 6.2: Hal Hardenbergh on RISCs, CISCs, and Fabs; Pink Noise; yet another VGA write mode; inside Turbo Pascal 4.0; the proposed ANSI C language standard; converting a Turbo C program to a TSR; 80286 and 80386 programming; 80386 software development; *The Norton Guide for BIOS/DOS Assembly.*

Vol. 6.3: Hal Hardenbergh on memory trends; Pink Noise; getting your consulting tools in order; Gordon Letwin on OS/2 memory management; virtualizing the 386; Pascal—a CASE of the jumps; Turbo Professional 4.0; BASIC—CALL, GOSUB, and multiline functions.

Vol. 6.4: Job shopping; windowing systems overview; reading VGA memory; X Window—a view from DOS; a digital clock in C for MS Windows; windowing in TopView, DESQview, and MS Windows; MS C 5.1 and the ANSI standard; a mouse unit for Turbo Pascal 4.0.

Vol. 6.5: Hal Hardenbergh on computers in the fast lane; Consulting—no-win jobs and too much work; VGA text paging; using dynamic arrays and records in Turbo Pascal 4.0; an OS/2 Presentation Manager application template; WATCOM C 6.0 and the ANSI standard; object-oriented OS/2 in C++; Ada and software engineering; REXX on the PC.

Vol. 6.6: PC DOS 4.0; OS/2 vs. UNIX; VGA screen saving and color selection; MASM quirks; addressing extra memory for DOS; DOS device drivers debugging and information utility; proposed ANSI C language standard; Ray Duncan on Forth; Ada data types and exceptions; PC floating-point hardware and software.

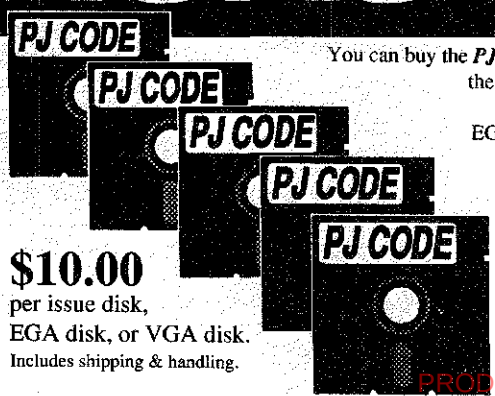
Vol. 7.1: GRAPHICS: High-resolution VGA color modes; graphics processors; modifying type constants stored in Turbo Pascal 4.0 and 5.0 compiled EXE code; comparing Pascal overlay managers; JPI's TopSpeed Modula-2; faster QuickBASIC code; Microsoft's CodeView debugger; Lattice C V3.3; smart terminal emulations in Windows; the COBOL-85 ANSI standard; Pink Noise—industry news and reviews.

Vol. 7.2: OPERATING SYSTEMS: undocumented DOS features; fast 80386 block moves under DOS; OS/2 dynamic link libraries; OS/2 multitasking; debugging utility for PM and Windows; EGA and VGA animation; disk formatting in Pascal; how to become a reseller; reviews of windowing and graphics tools, disassemblers, and *The 80386 Book.*

All mail orders must be accompanied by check or Money Order -- PREPAID ONLY.

CODE DISKS

Use the order form on the card opposite.
Order by phone with your VISA/MC.
CALL TOLL FREE
800/234-0386 ext 53



$10.00
per issue disk,
EGA disk, or VGA disk.
Includes shipping & handling.

You can buy the *PJ* code listings from volume 5.5 to the current issue. Code disks for the Michael Abrash "On Graphics" EGA/VGA series are also available.

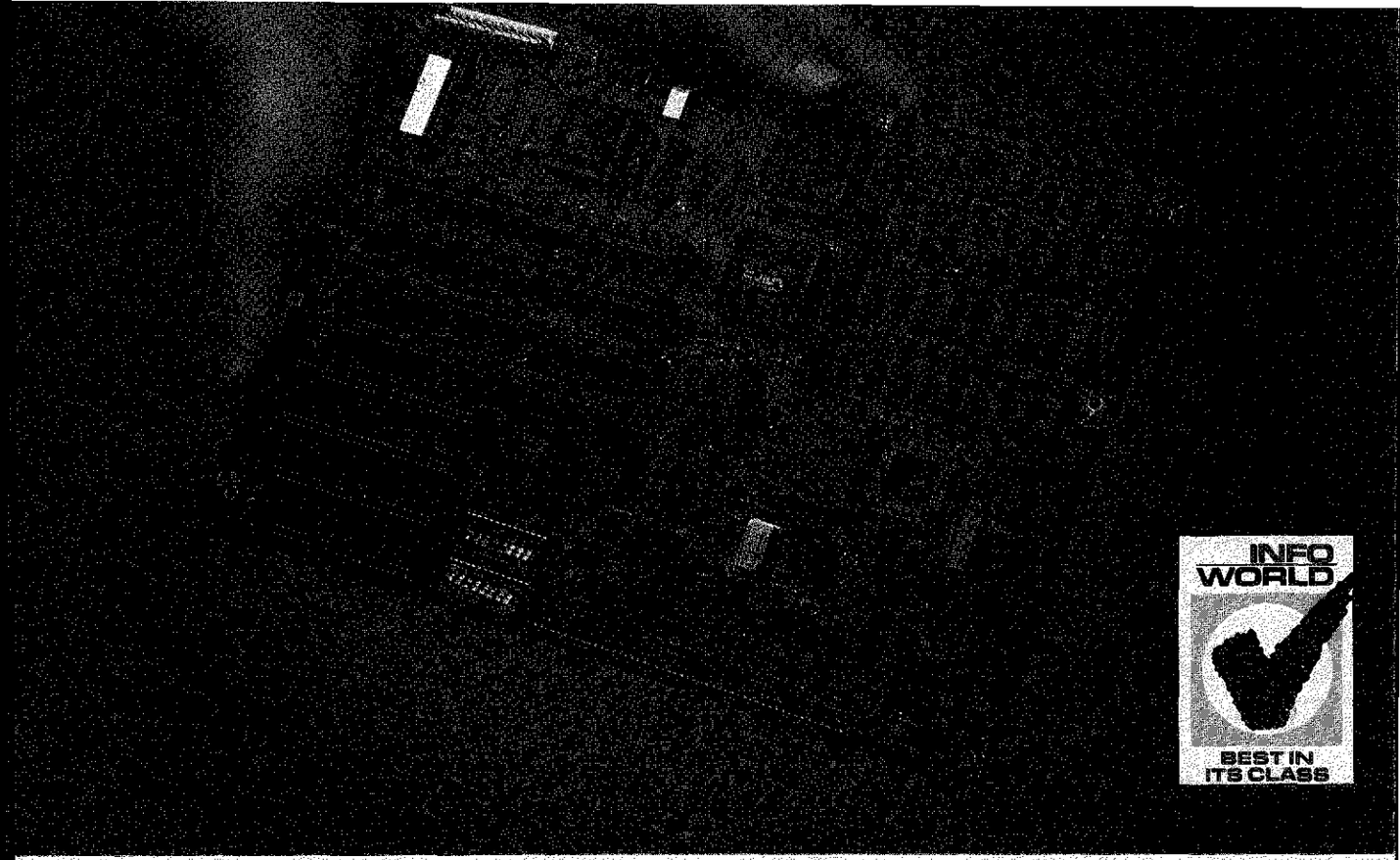
Diskettes are available for the following issues:

5.5	6.4	7.3
5.6	6.5	EGA
6.1	6.6	VGA
6.2	7.1	
6.3	7.2	

360K MS-DOS disks standard, other formats available.

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

PRODUCED BY OSSLIB.UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



INFO WORLD
BEST IN ITS CLASS

CONVERT YOUR PC, XT OR AT INTO A HIGHER FORM OF LIFE!

Don't let your PC give up the ghost — Hauppauge has arrived with a new spark of life: the 386 MotherBoard.™ Far more advanced than an accelerator card, our line of MotherBoards grace your PC, PC/XT, PC/AT or compatible with speeds equal to the IBM PS2 Model 80. And faster. Because we've built in 1 Megabyte of high speed RAM and a 387 math coprocessor socket for speeds that will make your PC glad to be alive!

UNIX/386 and OS/2 Compatible. To ensure a long, fruitful life, our 386 MotherBoards are compatible with the PC/AT (BIOS and I/O) — so you can run the new generation of DOS, OS/2. Our Boards also run Windows/386, UNIX V, PC-MOS/386, and XENIX/386. For *more* power, you'll find 16-bit expansion slots that accommodate the latest I/O expansion cards. Plus, we can add up to 15 Megabytes of 32-bit extended memory. No 386 accelerator card can give you so much versatility. **Only our 386 MotherBoards give your PCs an unlimited future!**

The Critics Applaud! *PC Magazine* awarded our Board "The Editor's Choice" for 386 Replacement Boards. *PC World* called it "the Upgrade Product of the Year." And *PC Computing* said: "The 386 MotherBoard is exceptionally compatible with existing IBM applications and peripherals — it will even run IBM's own version of OS/2."

Technical Features ▪ SX, 16, 20, 25, and 33MHz 80386 ▪ from 1 to 15 Mbyte of high speed 32-bit 4-way interleaved RAM ▪ PC/AT compatible BIOS and I/O system for support of OS/2 and UNIX ▪ 64K of 30 nsec static RAM cache on 25MHz and 33MHz boards ▪ Six 8-bit expansion slots and two 16-bit expansion slots (four on 386 MotherBoard for AT) ▪ One 32-bit expansion slot for up to 15 Megabytes of high speed memory.

386 MotherBoard for the PC, XT, AT and Compaq portable	$1595
386 MotherBoard 20, 25 and 33MHz for the PC/AT	from $1995
386 MotherBoard/SX for the mini chassis	$1295
32-bit RAM Boards	from $ 195
387 and Weitek coprocessors	from $ 500
16-bit combination hard disk/floppy disk controller	$ 175

Prices subject to change.

Our 386 MotherBoard will give your PC a new lease on life! For more information, call: **1 (800) 443-6284**. In New York: **(516) 434-1600**. All our products available at your local computer dealer. **(Ask about our GSA pricing!)**

Hauppauge Computer Works, Inc.
175 Commerce Drive,
Hauppauge, New York 11788
(In West Germany, call 0211-594-320)

Hauppauge!

Trademarks: IBM PC, XT, AT, PS2 and OS/2: IBM. Intel 386: Intel Corp.; Windows/386: Microsoft Corp.

Number 122 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

tools (profilers). Inside! is a profiling tool that will analyze a program in real time. Version 2.0 will provide timing information for functions, source code lines, fragments or blocks of code (called "events" by Paradigm), and DOS functions—and all with microsecond accuracy. The current list of high-level languages supported reads like a "Who's Who" in the DOS compiler world—Microsoft C, QuickC, FORTRAN, and QuickBASIC; Borland Turbo C and Turbo Pascal; WATCOM C; Lattice C; Logitech Modula-2 and JPI Modula-2; and Zortech C/C++. Each version costs $125.

A number of other profilers have been available under DOS, but Inside! sports two major distinguishing features. First, rather than sampling by churning up the PC's clock speed, Inside! accurately times functions and blocks of code execution by setting breakpoints (like a debugger), latching and reading the PC's clock count. This provides accuracy to about a microsecond. Second, Inside! uses the same debugging information used by the modern symbolic debuggers to locate functions and source line numbers. This debugging information is provided by your favorite compiler to fully automate the use of Inside!. To use Inside! effectively requires only that you set the appropriate compiler/linker switches so that this information is appended to the EXE file. The Lattice version uses the map file, and so it may be used with any compiler generating Microsoft-compatible map files. The Microsoft versions will function with any compiler that generates Codeview information (MetaWare, SLR, etc.).

Inside! supports a number of analysis options. The default is a function-timing mode that accurately tracks the count and time spent in each function invocation. Time spent in DOS may be included or excluded. A powerful line-timing mode will time the execution of source code lines in a single module or throughout an entire program. A MERGE utility will combine the line-timing results with the original source code into a single file for easier diagnosis. I found the arbitrary event-timing option particularly useful. Events are timed from a starting module:line number to some ending module:line number. Multiple event timings are supported. Other options include call target timing, noting nonexecuted functions or source lines, support for Microsoft overlays, and assembly language programs. Parameters may be provided on the command line or via an environment variable. A simple utility is also provided to time the load-and-execution time of any executable program.

Normally, the timing information is written to a disk file or the console. Timing information is given in both elapsed milliseconds (to the microsecond) and percentage of total time. Each version comes with a manual (75-plus pages) that notes the proper compiler/linker switches required along with specific features. The program overhead is about 60K but increases with the number of source lines or functions being timed. Some weaknesses: The number of events or functions to exclude/include is limited by the DOS command line and the environment variable, and the manuals lack indexes. Paradigm is currently working on a special version of Inside! for Microsoft Windows—clearly a much-needed product. Overall, I found Inside! to be an outstanding and easy-to-use performance-analysis tool, a very useful addition to your toolbox to fine-tune your programs.

ASMFLOW
Quantum Software
19855 Stevens Creek Blvd, Suite 154
Cupertino, CA 95014
408-244-6826

Quantum Software has released version 1.3 of ASMFLOW ($99.95), an assembly language flow-charting and source code–analysis tool. This is another of the unique and useful DOS products without any competition. ASMFLOW generates call tree diagrams and flow charts from 80X86 assembler source files. ASMFLOW can process a single file or a list of files (wildcards are supported) or even take the list from a makefile. ASMFLOW can be invoked in a full-screen menu mode with context-sensitive help or in a command line mode. The menu mode makes this product easy to use even by a novice. An INCLUDE variable in the environment can be used to locate any include files.

Aside from the more obvious call tree diagrams, ASMFLOW supports a number of useful options. The flow-chart option can generate diagrams of the program flow inside assembly language modules by visually showing loops, jumps, and nesting. Other features include determining maximum stack size, detecting recursion, procedure cross-reference, CPU register analysis in a procedure, instruction frequency histograms, CPU timing analysis, and flagging of questionable code. The CPU timing analysis will calculate the maximum and minimum numbers of clock cycles for the 8086/8087, the 80186/80187, the 80286/80287, the 80386/80387, or all of these CPUs in a single pass. These times are approximate, since they don't include the impact of the 80X86 prefetch queue, video memory wait states, and so forth. I found the tree diagrams and procedure register usage to be very useful indeed. ASMFLOW comes with a solid manual (56 pages) with an index.

ASMFLOW executed fast and was able to swallow large modules of assembly language (Kermit source code) without complaining. The program uses the Intel PROC and ENDP directives to locate functions and procedures. Older code without these PROC directives will be assumed to be one large in-line routine, for which ASMFLOW has some limits. The current liberal limits are 64K characters per procedure, 2,000 state-

FOR SERIOUS PROGRAMMERS ONLY

NEW RELEASE

OPUS MAKE

VERSION 5.2

The FASTEST most FULL-FEATURED make utility available.

Using OPUS Make will result in you spending less time doing compilations and other repetitive software tasks.

- DOS and OS/2 executables on the same disk.
- DOS version uses only 3K of memory!
- Read-time and Run-time conditional directives.
- Loop, Include and Error directives.
- Multiple directory support.
- Extensive macro capabilities.
- DOS version supports OS/2-like standard error redirection.
- POLYTRON PVCS, Burton Systems TLIB, LIB and SLR OPTLIB support.
- UNIX makefile syntax.
- Microsoft makefile compatibility mode.
- Also comes with OPUS MKMF: An automatic dependency generator that fully understands C preprocessor directives.

Both programs for only:

$89 + S&H

30-day money back guarantee.

For more information or to place an order call:

(415)664-7901

VISA/MC/cod/po

OPUS Software

1032 Irving St. #439, San Francisco, CA 94122

Serious software for programmers serious about their software.

Number 254 on the Reader Service Card

PRODUCED BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmer'sTools

from The Software Engineering Store & The Software Family

C

Borland Turbo C v2 Prof.	$169
Blaise C Tools Plus/5.0	$99
Creative Vitamin C	$169
Essential Graphics v2	$232
Faircom C-Tree/R-Tree Bundle	$519
Gimpel PC Lint	$99
Greenleaf Communications Lib	$179
Microsoft C v5.1	$299
Microsoft Quick C	$68
Novell Btrieve	$199
Softfocus Btree/ISAM *comprehesive file/index manager with all source*	$99
Wordcraft C Workshop *C training manual & software*	$69
Zortech C++ *Optimizing C++ Compiler*	$129

Turbo Pascal

Borland Turbo Pascal Prof.	$169
Turbo Power Professional v5	$99
Turbo Power B-Tree Filer ..... New!	$99

Other Languages & Tools

Digitalk Smalltalk/286	$164
Matrix Layout	$129
Microsoft Macro Assembler	$105
Microsoft Windows SDK v2.1	$328
Mortice Kern MKS Toolkit v2.3 *KORN shell & 130 UNIX™ -like utilities*	$175
Qualitas 386-Max *80386 memory manager*	$69
Software Masters Visual 80286 *interactive assembler training*	$89
Solution Systems Brief	$call

Visit our unique store for programmers in San Francisco!

Open manuals on display for most products

Free Catalog & Newsletter

We carry hundreds of products from dozens of publishers. Call or write for your free subscription to our catalog and newsletter.

Returns

We want you to pay for only the tools you really use. If you order anything featured in this ad (or most of the products in our catalog) you can open and use the product for up to 30 days and still return it if it doesn't meet your needs.

386 Hardware

Hauppauge 386XT replaces your XT motherboard to provide top 80386 performance with maximum compatibility. The 386XT works with virtually all 8-bit and 16-bit adapter cards and runs standard versions of 16-bit and 32-bit software including DESQview, Windows/386, OS/2 and UNIX. The board includes a 16Mhz 32-bit CPU, 1MB zero wait state memory, 80387 socket, and 8 slots (including 2 16-bit, AT, and 1 32-bit).

The 386XT is backed by a 30 day money-back guarantee. Made in U.S.A. One year warranty.

$1495

Hauppauge!

The Software Engineering Store

1-800-443-7176

In Calif: 1-800-635-0969

(415)957-0111

Visa • MasterCard
COD • Check

- Most orders shipped within 24 hours
- Free shipping on orders over $100

575 Mission Street, #600-PJ
San Francisco, CA 94105

The Software Family has a new name!

We've changed our name, but not our ownership, staff or outstanding service. The Software Engineering Store continues our tradition of fair pricing, expert technical support and honest business practices.

Logic Editor

Logic Gem *by Sterling Castle*: An electronic decision table structured like a spreadsheet to allow you to design a series of independent conditions and actions which describe your program's logic. It then completes the logic if your table is incomplete, generates a set of rules for you to edit, optimizes logic and generates your code. **$179**

Debugging

Periscope v4 *by Periscope*: The Periscope debuggers provide the combinations of hardware and software you need to solve really tough programming problems. Periscope includes all the debugging features you need, but with minimum overhead and maximum flexibility. With Periscope you can debug programs which need all memory, have real-time requirements and/or are picky about vector usage.

Periscope I w/512K — $675
0k memory overhead!

Periscope III — $1255
hardware breakpoints and trace-back for real-time applications

Technical Support

The Software Engineering Store carefully evaluates features, quality and support before offering a product to our customers. All of the products featured in this ad are backed by excellent technical support from the publisher — in many cases directly from the program developers. All are published by companies with reasonable update policies and with a track record for responding to customer requests for both bug fixes and new features. We further back the products with our own technical support. We are willing and able to intercede with the publisher if your needs aren't satisfied or to research problems caused by the interaction of products we recommend. We can also provide quotes for custom modification of source-licensed products, in-depth technical consultation or custom-developed products.

Prices subject to change without notice. ©1989 TSE. All Rights Reserved.

Number 98 on the Reader Service Card.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ments per procedure, 300 jumps or loops per procedure, and 64 simultaneous branches. This should easily handle most assembly language source modules. A new version, to be available in May, will eliminate these limits by inserting PROC directives and creating a new output file. Another indication of the value of ASMFLOW is that V Communications (Sourcer) and Wendin are both selling a version of ASMFLOW to complement their own products. With the advent of useful assembly language tools such as ASMFLOW and fast new assemblers, linkers, and librarians, we may yet see a new age for fast DOS and OS/2 programs written in assembly language.

FORTRAN Tools

Prism Toolkit Version 2.25
OTG Systems
P.O. Box 5250
Scranton, PA 18505-5250
717-343-8200

If you are porting any quantity of older FORTRAN code, the Prism Toolkit by Polyhedron Software (Great Britain) is a **must-have** product. The Prism Toolkit consists of a series of powerful tools for working with older FORTRAN source code. These tools include SPAG (a FORTRAN spaghetti code unscrambler), CRAM (automates documentation and overlay design), QMERGE (merges files with version selection), QSPLIT (splits out subroutines), CMAKE (a FORTRAN batch makefile generator), and LMAKE (a link file generator). SPAG is the primary utility and is without equal in the FORTRAN marketplace.

SPAG unscrambles spaghetti FORTRAN code by reordering blocks of statements in such a way that the logical structure is improved. For example, GOTOs can be replaced with FORTRAN 77 block IF and ENDIF statements. But that's only a small part of its capabilities. SPAG will optionally convert hollerith constants to character strings in FORMAT statements, convert source code to uppercase or lowercase, ensure terminators for DO loops, switch VAX-style (and soon FORTRAN 8X) DO .. ENDDO statements with FORTRAN DO .. CONTINUE statements (or vice versa), relabel statements, and reformat and indent code to show the basic structure.

About sixty options taken from an ASCII configuration file allow very complete control over the SPAG operation and the resulting output file. These extensive customization options allow users to adjust the style to personal tastes and local coding conventions. For example, individual control is available to convert to FORTRAN 77 block IFs separately from computed GOTOs, arithmetic IFs, and logical IFs. Conversion of source code between uppercase and lowercase is under separate control for symbolic variables, character strings, comments, and data declarations. Ten options control statement labeling. SPAG operates quickly and can automatically restructure and reformat older FORTRAN code at about 5000 lines per minute on a fast 386 machine. Compared with a programmer at maybe several hundred lines a day, SPAG has no competition. This is an impressive product for the FORTRAN market.

The other Prism tools are useful but are of more limited value. CRAM analyzes FORTRAN source code and can produce call summary charts and structure charts and can generate an optimal overlay structure suitable for direct input to PLINK86 under DOS or the IBM MVS linkage editor. CRAM can also check for argument count mismatches, recursive subroutine calls, and undefined main or subprograms. QMERGE combines a number of input files in alphabetical order into a single output file. QMERGE also allows selection between alternative versions identified by markers in the input file. QMERGE comments out unwanted lines and uncomments the selected version. Version selection uses *-IF, *-ELSEIF, *-ELSE, and *-ENDIF lines embedded in the input files. QSPLIT is a simple but useful utility for splitting merged files created by SPAG or QMERGE into separate source modules. Both of these other tools insert a header character sequence (by default, **==) before each program unit that QSPLIT utilizes for splitting.

Having a hard time getting into your windowing software?

Frustrated with the limitations of your windowing package?

ÆWINDOS is a high-powered set of tools designed to interactively create windows at light speed with both DOS and UNIX systems.

Written entirely in portable "C", it includes the complete source code and even make files, so changes are quick and easy.

Don't wait until you're caught in a bind, call us today. Modular, portable, and efficient, ÆWINDOS won't leave you out in the cold.

ÆSOFT

1-800-634-5494 BBS (303) 449-5714

* UNIX is a trademark of AT&T.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

CMAKE provides a simple alternative to the standard MAKE utility for FORTRAN source files. CMAKE automatically maintains a dependency file to keep track of FORTRAN INCLUDE directives and generates a batch file for recompilation. CMAKE has options for selecting the host compiler and compiler options to use. LMAKE is a simple utility for generating a list of object files in the current directory in the appropriate link response file format for Microsoft-compatible linkers or Phoenix's PLINK86. The Prism Toolkit comes with a brief but adequate 60-page manual, lacking an index.

Prism is distributed by OTG systems in North America. Since this is really a mainframe product also sold into the DOS market, the DOS pricing may seem high. The DOS version of SPAG/QSPLIT is $1200, and the entire toolkit is $1560. A 386 version is priced at $1800 (SPAG) and $2400 (Toolkit). The DOS version unscrambles up to 250 noncomment lines per program unit, while the 386 version handles 500-line subprograms. There are no limits on the overall size of your programs or the number of subroutines in a program to be restructured. The more costly minicomputer and mainframe versions (VAX, PRIME, etc.) have virtually unrestricted line limits—3000 lines. These line limits for the DOS versions are negotiable with OTG, based on price. Although by DOS standards this is a costly product, if you have 20,000 lines of older FORTRAN code, Prism is really a bargain.

Conferences and Exhibits

Software Development '89
Miller Freeman Publications
500 Howard Street
San Francisco, CA 94105
415-397-1881

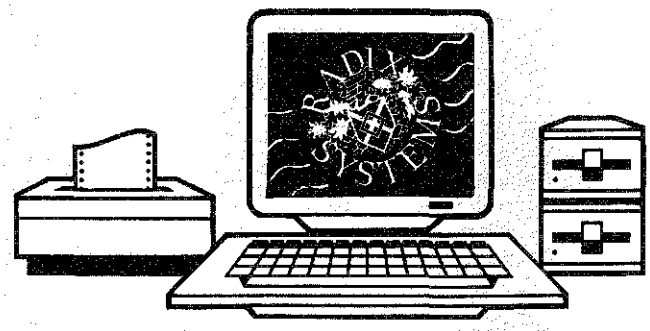
In February, Miller Freeman hosted their second annual programmer's workshop and exhibition. Although the course was not inexpensive, the range of material covered and the caliber of many of the speakers made this a worthwhile event for programmers. The exhibition (available under separate admission) is really the only software show focused on programming products and tools. Microsoft gave away copies of the IBM OS/2 1.1 EE (bundled with Borland's Sidekick for OS/2) to participants who qualified by collecting stickers from exhibitors. This may sound surprising, but it was actually a smart marketing decision to get OS/2 into the hands of more developers. Microsoft also endorsed Common View from Glockenspiel (marketed in the U.S. by Imagesoft), an object-oriented library and C++ translator for both Windows and Presentation Manager applications. Ironically, Common View uses a DOS extender from Rational Systems for the translator. At last, it's official that Lotus is also using Rational's DOS extender.

Borland's party featured a rather professional sounding jazz band made up of Borland employees and friends. Philippe Kahn and Eugene Wang traded some mean saxophone solos. Borland's "turbo debugger" (a pink fly swatter) was given to party goers.

I had a chance to talk directly with a number of developers and see products firsthand. I was most impressed with MMCAD's C development tools for the Mac, which are without competition. These tools illustrated for me the advantage of the windows metaphor for a programmer. Bill Summerlin (MMCAD) now does most of his development on a MacII, which is understandable, with LightSpeed C compiling at 40,000 lines per minute (the MMCAD DOS C tools are excellent as well). I also saw a number of products in the small-case-tools category—a hypertext editor and source code analyzer 4C (Tri-Technology Systems) and a nifty C help/database system, C. C. Rider (Western Wares). C. C. Rider compiles all of your C source code for a large project into a database. The database is usable from within your favorite editor to bring up function prototypes, include MACROS, and structure definitions at the touch of a hot key. The latest version of Instant C-16M (Rational Systems) functions like an incremental compiler with built-in hypertext editing, even though it's really an interpreter.

A number of products focused on Windows. NuMega Technologies showed their latest version of Magic-CV Windows, which allows debugging Windows applications using Codeview on an 80386 machine, providing the full 640K for your application code. Both Imagesoft and Architech showed versions of Sun's NeWS ported to OS/2.

Continued . . .



BRING DATABASE POWER TO YOUR SCREEN

Now you can achieve database flexibility in screen processing with the Radix Screen Management System. It's a language independent screen and keyboard controller. The package includes a facility that allows full screen editing of screen maps. Once created, a screen map is allocated and released at runtime, no compiling or linking necessary. Use the screen map in different applications developed under different languages. There are only eight commands to learn which control the screen maps. In addition, an Internal Document Facility will automatically display and scroll a window of your application's help text.

Other features include:

- **Window and Overlay Screens**
- **Alter Attributes at Runtime**
- ***Move Data Field-by-Field or in groups***
- **Request Only Modified Values**
- **Update the Screen Map Anytime**
- **Extensive Documentation**
- **Uses Standard Command Interface**

To order, send $198 plus $7 for shipping. No royalties.

Requires an IBM PC or compatible, 512KB of memory, and DOS 2.0 or higher. UNIX and OS/2 versions will be available soon. For more information call (201) 697-2630, or send $5 for demonstration diskette and Concepts & Facilities manual.

RADIX SYSTEMS INC.
P.O. Box 306
Rochelle Park, NJ 07662

Number **251** on the Reader Service Card

JAKE™: A NATURAL LANGUAGE C LIBRARY
Turns English Sentences into Function Calls & Data Structures
OUR CUSTOMERS USE JAKE FOR

COMMANDS	Script Languages Process Control Expert System Rules *Command Languages*
QUERIES	Hypertext Systems Help Systems Ad Hoc Database Queries Library Card Catalogue Retrieval
ANALYSIS	Electronic Funds Transfer Stock Market Tape Monitoring

JAKE $495. Interactive Demo $10
CALL (408) 438-6922 VISA, MC

English Knowledge Systems, Inc.
5525 Scotts Valley Dr. Suite 22
Scotts Valley, CA 95066

Usable with Microsoft C and Turbo C.
Context-sensitive semantic processing.
Easy interface to any application. Mem: 64K.

Number **243** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FINALLY!

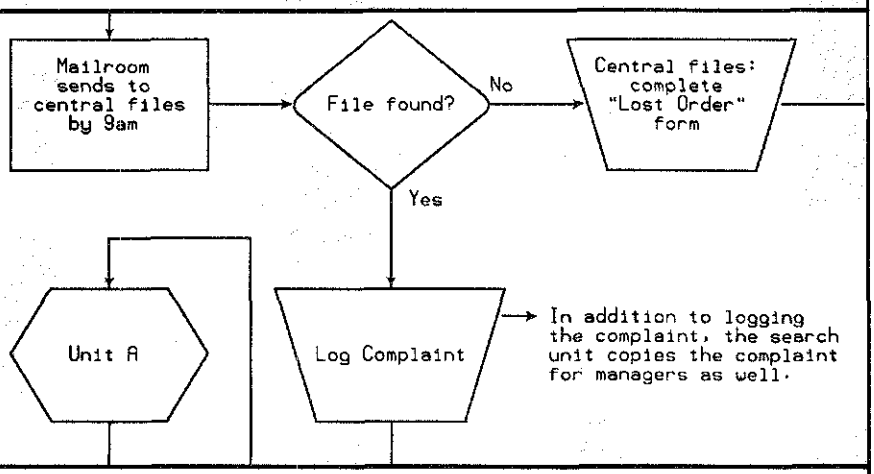
EASYFLOW

An on-screen flowchart processor that knows about flowcharts - not just another "screen draw" program that makes you do most of the work. EASYFLOW is a powerful full-screen graphics program dedicated to flowcharts and organization charts. With it you can quickly compose charts. More important, you can easily modify charts so they are always up to date.

- **Automatic:** Fully automatic text centering within shapes, both horizontally and vertically. Fully automatic line routing & re-routing.
- **Fast:** Written in assembly language for speed.
- **Large:** Charts up to 417 columns wide by 225 lines high. Chart too large for your printer? *EASYFLOW* automatically breaks the chart up & prints it in page size pieces.
- **Standard:** All standard flowcharting shapes included. Custom shapes can be ordered.
- **User friendly:** Don't take our word for it. PC Magazine says "*EASYFLOW lives up to its name. It's hard to imagine any easier and more flexible way to produce basic and even complex flowcharts*". [March 10th 1987 issue, page 278.]
- **Mouse:** Optional but fully supported.
- **It prints:** On most popular matrix printers including IBM, Epson, Toshiba, HP LaserJet, LaserJet-Plus and many others.
- **It plots:** On HP7440, 7475, 7550, 7570, 7585B and compatible plotters.
- **It works:** We are contractually prevented from mentioning the name of the "big eight" accounting firm that purchased a world-wide site license, but we can tell you that they spent months evaluating all available flowcharting packages before choosing *EASYFLOW*.
- **Rush delivery:** Order by noon today (eastern time) and you'll have it tomorrow. Rush delivery charge is $10.50 (instead of $2.00) and is available only to continental USA and Canada.
- **Documented:** 100 page manual plus over 150 screens of context sensitive help.

EASYFLOW works on IBM PC's, IBM PS/2 and compatibles. Requires 384 K memory, DOS 2.0 or higher and an IBM CGA/EGA/VGA or Hercules monochrome compatible adapter card.

Order direct for only **$149.95** + $2.00 S&H (USA/Canada), $10.00 (foreign). Payment by M.O., cheque, VISA, Mastercard or Company PO.


Mailroom sends to central files by 9am
File found?
No
Central files: complete "Lost Order" form
Yes
Unit A
Log Complaint
In addition to logging the complaint, the search unit copies the complaint for managers as well.

The chart fragment above was produced on an HP LaserJet-Plus and is actual size and unretouched. Publication quality charts like this can be produced using only minutes of preparation time and seconds of print time.

HavenTree Software Limited
PO Box 1093-V
Thousand Island Park, NY 13692

Order Desk: 1-800-267-0668
Information: (613) 544-6035 ext 31
Telefax(G3): (613) 544-9632

Number **154** on the Reader Service Card

Books

***A Guide to the SQL Standard,* by C. J. Date**
Addison-Wesley Publishing Company
Route 128
Reading, MA 01867
617-944-3700

Clearly, relational databases are descending onto the PC platform, and the Structured Query Language, or SQL, (originally developed by IBM) has been adopted as the standard language for dealing with them. However, the official ANSI SQL standard documents are not intended for the casual reader. In this book, Chris Date does an excellent job of providing a comprehensive yet very readable view of standard SQL. The emphasis of this book is on the official standard version of SQL instead of one of the implemented dialects. Date also emphasizes the use of SQL for programmed (*embedded* applications) as opposed to *interactive* access. Nowadays, with the computer-book presses churning out new titles overnight, it is truly a pleasure to pick up such a well-written piece. The author is not only a database scholar but also a fine wordsmith.

Date provides a very thorough treatment of SQL, discussing all language aspects in detail. Date goes beyond the standard by providing interesting comments and critiques in the context of various SQL features or omissions. He also discusses the differences between standard SQL and the IBM version supported by DB2 and examines official proposed extensions to the standard. A number of examples and sample problems are used well for illustration.

Specific chapters cover basic data objects and language elements, data definition and schema, the module language, cursor operations, views, common language constructs, embedded SQL, and extensions. Appendices detail the SQL grammar, language levels and conformance, differences from DB2, and an annotated SQL critique. The author has written several other titles on specific implementations of SQL that are also excellent, but this book will serve the reader as both a fine tutorial and a reference guide to SQL.

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BASIC to C

BAS C v3.2 accepts BASICA, Quick BASIC, CBASIC86, or most other BASICs by adding new statements, functions(Language Extender). Generates structured, indented, scoped MS/Turbo/Lattice/Aztec C. All Memory model, compile option supported. Runs on MS-DOS, XENIX, UNIX. Demo disk. From $199.

GOTOLESS CONVERSION
P.O. Box 835910
Richardson, TX 75083
214-404-1404 **RS# 127**

YOUR CUSTOMERS DESERVE THE BEST - CUT YOUR SUPPORT COSTS

Install your software as if you were there yourself, adapting to hardware, software, environment and installer wishes. Just "fill in the blanks" to create multiple adaptable options in minutes, intelligently controlling AUTOEXEC, CONFIG and much more. New installations, updates, MODS. 250-page User's Guide, 45 pages on automatic handling of special and abnormal situations.

INSTALLATION TOOLKIT ver. 3.0, $99.00. No copy protection/royalties, money-back guarantee, VISA/MC.

Helpful Programs, Inc.
P.O. Box 16078
Huntsville, AL 35802-1627
800-448-4154 **RS# 280**

MAKE MONEY WITH YOUR PROGRAMS

New guide shows you how to put your software in the shareware market. Remarkably easy steps lead to profitable earnings. Examples, tips and tricks shown, plus addresses to 60 free distribution companies! Also covers pricing, cover letters and customer support. This is the definitive how-to for free-lance programmers.

Shareware Guide only $10, example disk $10 extra.

Apogee Software
4206 Mayflower
Garland, TX 75043 **RS# 273**

PL/M TO C TRANSLATOR

LISP-based translator for major assistance in translating from INTEL PL/M to C language. Human readability a high priority. Sample listings available. Runs under MS-DOS 2 and up. Ports possible for VAX with VMS or BSD 4.2.

$5000 site license includes 3 months of telephone support.

Frank Korzeniewski Consulting
1564-A Fitzgerald Drive #137
Pinole, CA 94564
415-799-1819 **RS# 209**

AT LAST! A C SOURCE CODE ANALYSIS TOOL WITH FULL PREPROCESSING CAPABILITY

C:LINES/C:TREE features

- ANSI draft compatible preprocessor
- Outlined listings showing flow of control within procedures
- Tree diagrams showing flow of control between procedures
- Source code reformatter

$79.95, *shipping $3 domestic*, $15 overseas. MD residents add 5% sales tax. Requires PC compatible, 256K.

SoftRex
4807 Bethesda Ave. #287
Bethesda, MD 20814
301-881-8274 **RS# 185**

DOCUMENTATION COMPILER

EASILY CREATE AND SHARE SOURCE CODE DOCUMENTATION. DocuMotion builds cross-indexed project documentation libraries from your source code comments. Language-independent. Resident/nonresident view utility gives instant shared access to all project documentation, manuals, etc. powerful print and copy capabilities. Folding documents, electronic bookmarks, search, LAN-compatible, more.

Personal version $149, 3 station network $359, 6 station network $597.

NWP-Intelligent Solutions Inc.
P.O. Box 20478
Bloomington, MN 55420-0478
612-884-5860 **RS# 283**

C SOURCE CODE
FOR ACCOUNTING BUSINESS AND MANUFACTURING MANAGEMENT
XLN *offers K&R C source code* modules for general ledger, sales and accounts receivable, purchasing and accounts payable, inventory, bill of materials, MRP, CRP, PERT, maintenance management, personnel and payroll, and more.
XLN SOFTWARE
3680 Tuggle Road
Buford, GA 30518
800-543-5770
FAX: 404-859-0051
GA, Int'l or support: 404-945-8745

Number 237 on the Reader Service Card

PJ CODE IS AVAILABLE
ON DISK—CALL
800-234-0386
$10.00 each, VISA/MC
See ad on page 85 for more information on BBS downloads.

Turbo Taskforce.

Want to write better code in less time? We're the Turbo User Group, and since 1984 we've been helping people do that and more with all of Borland's language products. Membership in TUG will net you -

- A subscription to TUG Lines, the group's bi-monthly *journal, containing utilities, bug reports, patches,* workarounds, product reviews and more
- Access to TUG's "Troubleshooters" - volunteers who help solve programming problems by phone
- A discount on The TUG Boat, the group's multi-user BBS, where you can contact, correspond and conference with fellow users, participate in language SIGs and download "hot" Public Domain software
- Access to TUG's extensive disk library of hand-selected and tested Public Domain and ShareWare utilities, routines and applications
- And more!

Don't just work *harder* - join with us in our commitment to work *smarter!*

Turbo User Group
Post Office Box 1510 • Poulsbo, Washington 98370
(206) 779-9508
Annual dues (US funds): U.S. $24.00; Canada $28.00; Overseas $39.00

Number 259 on the Reader Service Card

PC-lint
C Programmers: Locate bugs, glitches, quirks and inconsistancies in your C code. PC-lint supports multiple modules, K&R, ANSI. Specify OS/2 or MS-DOS. Only $139.
Gimpel Software
(215)584-4261
3207 Hogarth Ln
Collegeville PA 19426
30 Day Money-back Guarantee

Contact this advertiser directly.

LATEST AWARD BIOS
Update Your PC/XT • 286 • 386
Support for:
- **Enhanced Keyboards**
- **EGA & VGA Graphics**
- **Custom Drive Tables**
- **3.5" Floppies & More**

Authorized AWARD Distributor
(800) 423-3400
KOMPUTERWERK, INC.
851 Parkview Blvd.
Pittsburgh, PA 15215

Number 155 on the Reader Service Card

D-ISAM
Industry standard file manager. Only $145 for DOS libraries, $595 for UNIX/DOS source code.
'W'
UNIX/DOS character windowing. $95 for DOS libraries, $295 for UNIX/DOS source code.
BYTE DESIGNS LTD.
#6-3511 Viking Way
Richmond, B.C.
V6V 1W1 CANADA
(800) 663-8547
(604) 278-5200

Number 219 on the Reader Service Card

MIDNIGHT ENGINEERING™
A new magazine for entrepreneurial hardware and software engineers that will encourage and challenge you to personally develop and market your own products.
- **PRACTICAL ARTICLES**
- **INSIGHTFUL INTERVIEWS**
- **DETAILED PRODUCT REVIEWS**

This is your chance to influence a brand-new magazine. *PJ* readers are invited to a FREE copy of the premiere issue of *Midnight Engineering* by calling or writing us with ideas.
DATArx
111 E. Drake Rd., Suite 7041
Fort Collins, CO 80525
303-223-2120

Number 272 on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
282	Acucobol Inc.	47
220	Aesoft Corporation	92
273	Apogee Software Productions	95
30	The Austin Code Works	79
207	Automation Education	85
51	Az-Tech Software, Inc.	41
244	Block Island Technologies	68
3	Borland International	C4
219	Byte Designs, Ltd.	95
284	CNS, Inc.	68
281	COBRA Systems Software	46
277	Covox Inc.	33
255	The C-Team, Inc.	69
272	Datarx	95
229	Disk Software, Inc.	29
241	Emerald City Software	8
243	English Knowledge Systems	93
267	Falk Data Systems	67
209	Frank Korzeniewski Consulting	95
274	Genesis Data Systems	7
261	Genus Microprogramming	29
*	Gimpel Software	95
204	Guidelines Software, Inc.	83
122	Hauppauge Computer Works, Inc.	89
154	HavenTree Software Limited	94
258	Helios Software	6
280	Helpful Programs	95
226	Ithaca Street Software, Inc.	67
211	KADAK Products Ltd.	41
155	Komputerwerk	95
63	Laboratory Microsystems Incorporated	41
44	Lahey Computer Systems Inc.	7
246	Logical Alternatives Incorporated	33
271	Lugaru Software Ltd.	31
212	Magna Carta Software	20
233	Mansfield Software Group	7
223	Marietta Systems	71
143	Metagraphics Software Corporation	87
45	MetaWare Incorporated	1
232	Microsoft Corporation–Human Resources	9
234	Microsoft Press	35
104	Microway	49
121	MISOSYS, Inc.	45
238	MISOSYS, Inc.	77
266	Mix Software	81
94	MMC AD Systems	50
109	Nanosoft	51
278	Netwise	C3
160	Nu-Mega Technologies	3
283	NWP Intelligent Solutions	95
16	Opt-Tech Data Processing	17
254	OPUS Software	90
270	OTG Systems Inc.	33
*	Paradigm Systems	14
208	PC Tech	27
6	The Periscope Company	19
*	***PJ Code DiskScription***	8
*	***Programmer's Journal*** Mail Order Store	88
228	ProTech Marketing, Inc.	37
97	Qualstar Corporation	14
252	Quantum Software	29
147	Quarterdeck Office Systems	C2
251	Radix Systems	93
222	Rainbow Technologies	23
20	Rational Systems, Inc.	21
198	The Research Group	13
148	SEMWARE	62
269	Sequiter Software Inc.	71
165	SLR Systems	38
33	Soft Advances	15
185	SoftRex	95
275	Software Development Technologies	77
98	The Software Engineering Store	91
221	Software Security	5
268	Sterling Castle	22
158	Summit Information Systems, Inc.	81
253	Sutrasoft	29
166	The Tool Makers	40
262	Trio Systems	63
276	Tri-Technology Systems Inc.	77
25	TurboPower Software	51
259	Turbo User Group	95
125	V Communications, Inc.	38
100	Western Wares	69
73	The Whitewater Group	70
237	XLN Software	95

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers with an * prefer to be contacted directly. ***PJ***'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors.

COMING ISSUES

July/August

Networking and Communications:

Communications in Windows/PM *by Bill Hall*

Network Protocols *by Steven Baker*

OS/2:

OS/2 Device Drivers *by Ray Duncan*

PJ Forum:

Pink Noise *by Frank Greco*

September/October

Object-Oriented Programming

Advertise in *Programmer's Journal*

July/August 1989 Volume 7.4

Space Reservation Deadline:

May 5, 1989

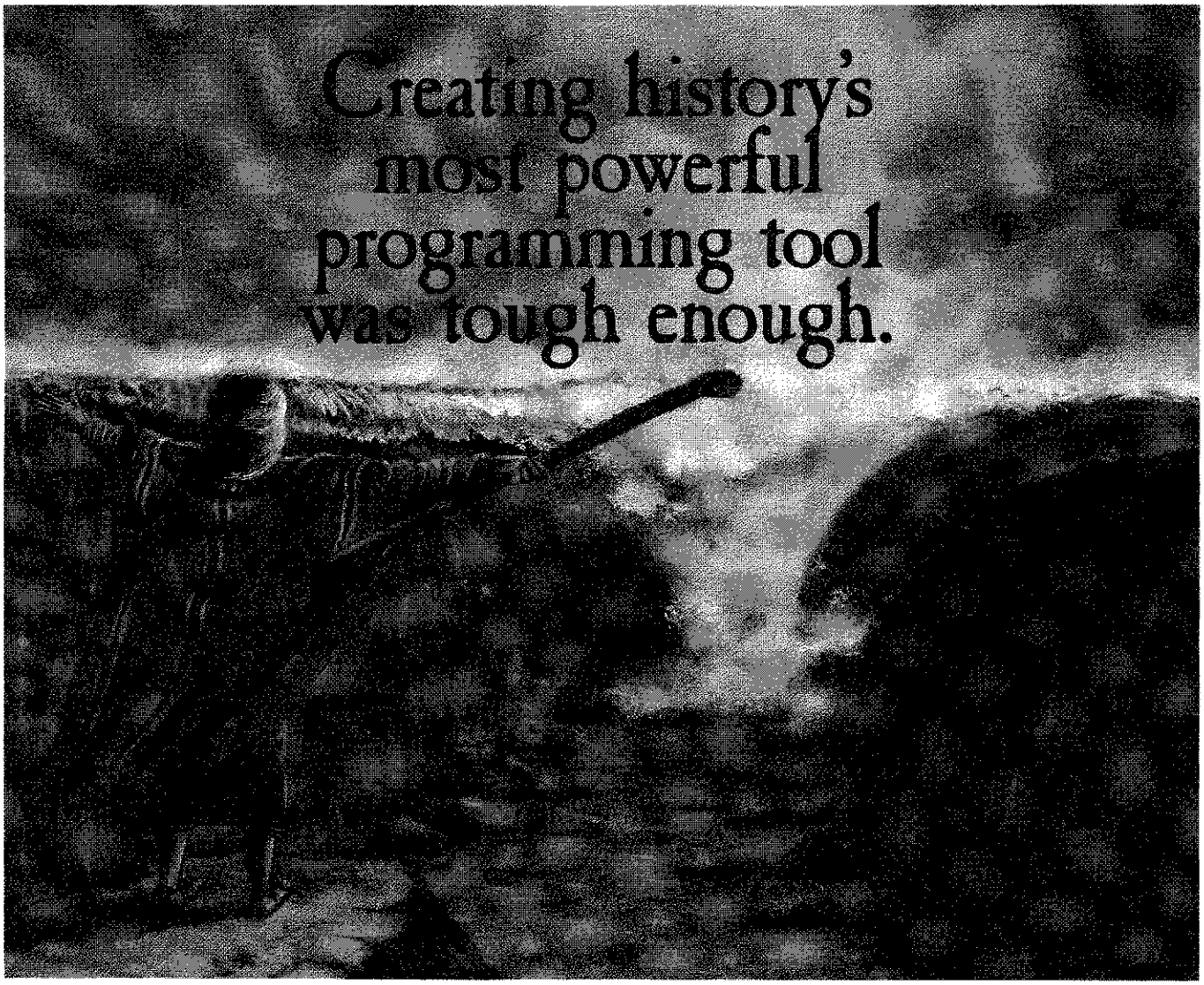
Materials Due:

May 19, 1989

Please call *Programmer's Journal* to reserve your space.

(503) 747-0800

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Getting programmers to believe it could take a miracle.

We know you've heard it before. Products claiming breakthroughs of biblical proportions that are nothing more than complete washouts. And while our product is for real, we don't expect you to buy the hype. Yet.

Here's the promise. For the wide range of hardware, operating systems and networks we support, our software provides your applications with all the communications routines you need to achieve transparent, trouble-free network computing.

The Netwise® RPC TOOL™ helps you build enterprise-wide software by generating machine- and operating system-specific C source code. We also provide libraries of network interface routines, so you can easily port your applications to an impressive variety of network environments. With Netwise, you control the full potential of your networked applications. Now and in the future.

How we do it requires more explanation than we have room for here. So we ask that you call for our free educational diskette and technical information kit. Or, order our RPC TOOL for a 30 day evaluation. If you still don't believe it's a modern day miracle, we'll refund your money. And if that's not true, may we be struck by lightning.

Don't believe us? Call us. 303/442-8280

NETWISE

2477 55th Street, Boulder, Colorado USA 80301

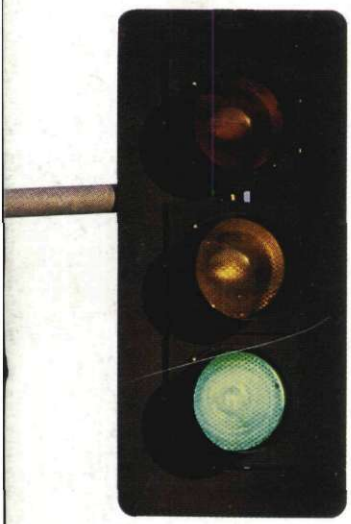
Number **278** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Pull out all the stops

Turbo C® Professional is the *only* production-quality C compiler with a completely integrated environment.

Everything you need—*all* the tools—are *included* in this environment, so you never waste time stopping, starting, and switching between tools.



And you're not forced into trade-offs between high-productivity programming and small, fast, reliable code. Instead you get the fastest and the best of both worlds.

There's tight integration between editor, compiler, linker, and debugger that lets you race through your program with a fast edit/compile/run/debug cycle.

Only Turbo C Professional gets it all together

Now everything you need to write *and debug* production-quality, optimized code in *both* C and Assembly language is all yours in one package.

With Turbo C Professional you get:

- Turbo C 2.0 with its own integrated development environment—Compiler, Editor, Debugger, and Linker.
- Plus a separate command-line C Compiler
- Turbo Assembler®—a complete Macro Assembler that's more compatible with MASM than MASM is with itself.
- And the new source-level Turbo Debugger® that lets you debug *any size program*. Turbo C Professional has it all.

Turbo C 2.0	Turbo Assembler	Turbo Debugger
■ Compiles over 16,000 lines per minute	■ Assembles up to 48,000 lines per minute	■ Debug any size program
■ Hypertext online Help	■ Compatible with MASM 4.0, 5.0, 5.1	■ Browse through structures with data debugging
■ Supports inline assembly	■ Full 386 support	■ Set conditional breakpoints, break on memory access
■ All six memory models supported	■ Assembles multiple files	■ Stop, run code, log expressions
■ More than 450 library functions		■ 386 ICE capabilities

Turbo C, Turbo Pascal, Turbo Assembler, and *Turbo Debugger* are registered trademarks of Borland International. Copyright ©1989, Borland International, Inc. All right reserved. BI 1286

CODE: MC08

Turbo Debugger is a winner

Turbo Debugger won *PC Magazine*'s most recent Award for Technical Excellence, and here's what they said:

"Everyone who's tried the Turbo Debugger agrees. It wins the (development tool) category's award for Technical Excellence hands down. The user interface is simple yet elegant; the program works the way programmers want to work. Yet again, Borland has advanced the state of the art in an eminently useful way."

Bill Machrone, Editor-in-Chief, *PC Magazine*

Debug *any size program*

Turbo Debugger lets you debug on a remote machine. That's a win. And in virtual mode of the 386, it allows you to debug *any size program*. Even your largest—*especially* your largest. That's a *huge* win.

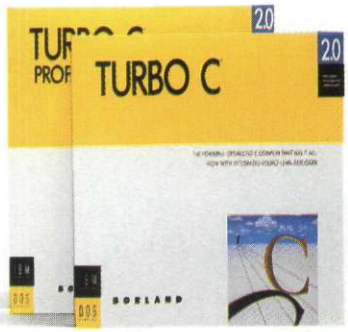
And it can give you 12 *different* views of your code. It supports browse-through data debugging; offers flexible breakpoints; supports in-circuit emulation; offers EMS support; has a "Point & Shoot" integrated debugging environment, and is completely CodeView® compatible. Turbo C Professional does all that, so *it* wins—and so do *you*.

Pull out all the stops

Turbo C's integrated environment lets you completely stop stopping. Your program is never interrupted. No stops and no g aps. You compile faster. Link faster. Work faster. Think faster. So turn to Turbo C Professional: There's nothing stopping you now.

This is no ordinary Demo

Not one, not two, but three different demos on the same disk: Turbo C, Turbo Pascal,® and Turbo Debugger. We'll send you the demo disk and fact sheets with technical information for $4.95. **To order, CALL (800) 345-2888 Ext. 200**



BORLAND

Number **3** on the Reader Service Card

PRODUCED 2004 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED